Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2008

Mauro André
Moreira Rodrigues

Unidade de Processamento e Sistema de Visão para um Robô Humanóide


Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática,
2008

Mauro André
Moreira Rodrigues

Unidade de Processamento e Sistema de Visão para um Robô Humanóide

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Prof. Dr. Filipe Miguel Teixeira Pereira da Silva, Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Prof. Dr. Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri

presidente

Doutor Paulo Jorge dos Santos Gonçalves Ferreira
Professor Catedrático da Universidade de Aveiro

vogais

Doutor Alexandre José Malheiro Bernardino
Professor Auxiliar do Instituto Superior Técnico da Universidade Técnica de Lisboa

Doutor Vítor Manuel Ferreira dos Santos
Professor Associado da Universidade de Aveiro

Doutor Filipe Miguel Teixeira Pereira da Silva
Professor Auxiliar Convidado da Universidade de Aveiro

agradecimentos / acknowledgements

Este objectivo não teria seria alcançado sem o apoio de algumas pessoas a quem gostaria de agradecer.

Em primeiro lugar, aos meus orientadores, Prof. Dr. Filipe Silva e Prof. Dr. Vítor Santos, pelo apoio e conselhos dados ao longo da realização deste trabalho.

A todos os elementos do Laboratório de Automação e Robótica, que me acolheram nestes meses e com os quais criei laços de amizade. Um abraço especial para o Miguel Oliveira e para o Eduardo Durana pela paciência de me aturarem desde o início do trabalho.

A todos os elementos, jogadores e treinadores, da equipa de Voleibol da AAUAv com os quais convivi no período 2004-2008. Um cumprimento especial para o Eurico Couto, uma das melhores pessoas que conheci, um verdadeiro "Capitão".

Aos amigos que infelizmente seguiram caminhos separados e que com alguns dos quais o contacto se foi perdendo: Pedro Azevedo, Ricardo Gomes, Ricardo Leandro e Sara Silva. Podem estar longe mas não estão esquecidos.

Aos amigos que partilharam desta etapa da minha vida: Adriana Abreu, Alexandre Vieira, Armando Cavaleiro, Cátia Ferreira, César Rodrigues, Fábio Gaspar, Pedro do Mar, Ricardo Rocha, Tiago Roque. Um grande abraço!

Aos meus avós, pelo apoio incondicional. O vosso orgulho em mim deixa-me muito feliz.

Ao meu irmão, pelos momentos de descontracção ao fim-de-semana e pelas conversas e discussões muitas vezes sem sentido. Acredita, ajudavam imenso a descontrair de uma semana de trabalho.

À Filipa, a minha namorada, pelo apoio que me deu ao longo destes últimos três anos. A sua presença no dia-a-dia foi muito importante para o meu sucesso.

Aos meus pais, por tudo. Isto nunca teria sido possível se não fosse por eles. Espero um dia conseguir retribuir esta oportunidade que eles me deram.

palavras-chave robô humanóide, sistema embebido, arquitectura distribuída, visão por computador, controlo visual

resumo Este trabalho descreve a integração da Unidade Central de Processamento, um computador embebido, numa plataforma humanóide e o desenvolvimento do sistema de visão do robô. É abordado o processo de alteração da estrutura da plataforma para a integração física, e também a configuração e implementação do ambiente de desenvolvimento por forma a permitir a integração numa arquitectura de controlo distribuída já existente. O sistema de visão é baseado numa unidade *pan-tilt* que movimenta uma câmara para aquisição de imagem. A informação retirada da imagem adquirida é processada e usada para fazer o seguimento de um objecto. Para o seguimento são usados dois algoritmos de controlo baseados na imagem.

keywords humanoid robot, embedded system, distributed architecture, computer vision, visual servoing

abstract This report describes the integration of the Central Control Unit, an embedded computer, on an humanoid platform and the development of the robot's vision system. The necessary changes on the physical support are shown as well as the configuration and implementation of the development environment, in order to allow the integration with the existing distributed architecture. The vision system is based on a pan and tilt unit supporting a color CCD camera for image aquisition. The visual tracking is based on the features of the acquired and processed image. Two different image-based algorithms are used for control.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Muitos foram os que sonharam, enquanto crianças, com a possibilidade de terem um robô humanóide capaz de se mover, agir e interagir como um humano. Este sonho tornou-se real para alguns investigadores [1, 2, 3] e promete também sê-lo para um conjunto alargado de entusiastas, nomeadamente cientistas e engenheiros. A área da robótica humanóide apresenta uma dinâmica crescente em resultado dos avanços tecnológicos verificados nos últimos anos. O envolvimento de várias companhias, sobretudo Japonesas, culminou com o aparecimento de várias plataformas, com destaque para os emblemáticos ASIMO da Honda e QRIO da Sony (Fig. 1.1).

(a)

(b)

Figura 1.1: Plataformas Humanóides (a)Honda ASIMO e (b)Sony QRIO

Estas empresas apresentam actualmente humanóides com alto nível de desenvolvimento e desempenho, que são capazes de andar, subir e descer escadas e inclusivamente correr com grande estabilidade. Níveis de desempenho como estes são alcançados devido aos enormes recursos económicos das empresas envolvidas. Intuitivamente pode-se inferir que objectivo maior passa pela completa autonomia destes humanóides quer em termos energéticos como em termos de decisão. Adicionalmente, é expectável que os humanóides possam ainda ser capazes de promoverem a auto-manutenção (*e.g.*: auto-recarregarem as baterias), aprender por si e adaptarem-se a novas situações, evitarem situações de perigo para os humanos, o meio envolvente e para si próprios e ainda interagirem em segurança com os humanos e o meio ambiente [31, 32, 33]. As actuais tendências da electrónica e sistemas embebidos tornam

este objectivo possível com *hardware* desenvolvido à medida das necessidades, sendo que o verdadeiro desafio centra-se então na utilização de componentes comerciais (*"off-the-shelf"*) [4, 5].

Muitos grupos de investigação, nomeadamente Universidades e laboratórios, iniciaram a construção de robôs de pequenas dimensões e baixo custo no sentido de realizarem pesquisa em áreas tão diversas como o controlo, a percepção, a navegação, o comportamento ou a cooperação. Este foi, também, o móbil principal que levou um grupo do Departamento de Engenharia Mecânica da Universidade de Aveiro a encetar a tarefa de construção de uma plataforma humanóide [6] (Fig. 1.2).

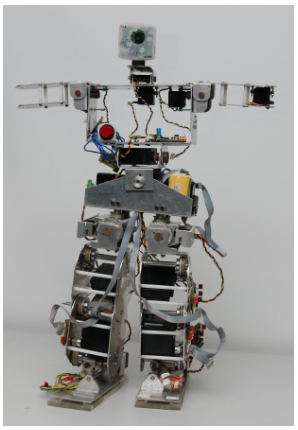


Figura 1.2: Humanóide com 22 graus de liberdade

O projecto PHUA (Projecto Humanóide da Universidade de Aveiro), iniciado em 2003, em estreita colaboração com o Departamento de Engenharia Electrónica, Telecomunicações e Informática, levou ao desenvolvimento de um protótipo que inclui o desenvolvimento de toda a estrutura mecânica [7, 8, 9, 10, 11], o *hardware* de arquitectura distribuída de controlo [12], alguns sensores de força e um sistema de controlo relativamente complexo para os servomotores que actuam nas juntas [13]. Este projecto tem como objectivo principal a criação de uma plataforma de investigação de grande valor pedagógico face aos enormes desafios científicos e técnicos e à diversidade de problemas associados. Ao mesmo tempo, pretende-se promover o envolvimento de estudantes da UA em competições robóticas internacionais como, por exemplo, o *RoboCup*.

O *RoboCup* é uma competição internacional anual [28] que visa o desenvolvimento da Inteligência Artificial (IA) [29, 30] e da Robótica criando um problema comum - até 2050, construir uma equipa de 11 robôs humanóides completamente autónomos capazes de ganhar à equipa de futebol campeã do mundo. Obviamente, desenvolver um robô capaz de jogar futebol é um enorme desafio. No entanto, oferecendo um desafio publicamente interessante, a competição consegue promover a pesquisa e servir como bancada de testes para tecnologias desenvolvidas pelas equipas para superar os desafios impostos (Fig. 1.3). Isto faz do *RoboCup* um projecto cooperativo e aberto à escala mundial, na área da robótica e IA (Fig. 1.3).

Uma caraterística singular é a tentativa sistemática de promover a pesquisa através de um domínio comum, com ênfase no futebol, existindo ainda outras categorias na competição como outros escalões de futebol robótico, busca e salvamento, dança e tarefas domésticas.

É também, talvez a primeira a assumir o objectivo de ganhar à equipa humana campeã do mundo. Construir um robô capaz de jogar futebol será certamente um resultado extraordinário na área da robótica e inúmeros projectos derivados são expectáveis ao longo do decorrer do projecto. Este tipo de projectos são chamados de "projectos de referência" e o *RoboCup* é definitivamente um projecto desse tipo.

Figura 1.3: Robôs Humanóides presentes na edição RoboCup 2007

É objectivo primeiro deste trabalho proceder à integração, na plataforma, da Unidade Central de Processamento de modo a dotar o Humanóide de capacidade de processamento "on-board" e assim conferir-lhe autonomia. É abordada a escolha da Unidade de Processamento e do sistema operativo a instalar, bem como a implementação do ambiente de desenvolvimento e configurações necessárias à optimização do desempenho do conjunto. É também objectivo deste trabalho, o desenvolvimento do Sistema de Visão do Humanóide com foco no seguimento de objectos. O seguimento de objectos é efectuado por uma câmara montada numa unidade *Pan-Tilt* (PTU), o pescoço do Humanóide, com auxílio do grau de liberdade, no plano sagital, do tronco.

O documento encontra-se organizado da seguinte forma: o Capítulo 2 descreve as características globais da Plataforma Humanóide desenvolvidas em trabalhos anteriores e as alterações efectuadas no âmbito deste trabalho. O Capítulo 3 aborda a integração da Unidade Central de Processamento, criação do ambiente de desenvolvimento e repectivas configurações. No Capítulo 4 é discutido o Sistema de Visão, passando por diferentes abordagens ao algoritmo de *tracking* até ao método proposto para a inclinação do tronco, complemento da PTU. No Capítulo 5 são apresentadas as conclusões do trabalho desenvolvido e perpectivas de trabalho futuro.

Capítulo 2

A Plataforma Humanóide

2.1 Enquadramento e Motivação

A plataforma é um Robô Humanóide desenvolvido na Universidade de Aveiro. O projecto, iniciado em 2003, levou ao desenvolvimento, de raíz, de um protótipo que inclui a criação de toda a estrutura mecânica, o *hardware* e *software* da arquitectura distribuída de controlo, alguns sensores de força e um sistema de controlo, relativamente complexo, para os servomotores que actuam nas juntas.

A plataforma contém ainda capacidade de processamento e aquisição de imagem para interacção com o meio envolvente. A Unidade Central de Processamento é um computador embebido, que será descrito mais detalhadamente no Capítulo 3.

As principais características da plataforma são: 22 graus de liberdade, cerca de 70cm de altura, uma massa total de 7kg e arquitectura distribuída de controlo. Os 22 graus de liberdade presentes na plataforma encontram-se distribuídos da seguinte forma (Fig. 2.1):

- 2 na cabeça (2×1) (PTU)
- 3 em cada braço (3×2)
- 2 no tronco (2×1)
- 3 em cada anca (3×2)
- 1 em cada joelho (1×2)
- 2 em cada pé (2×2)

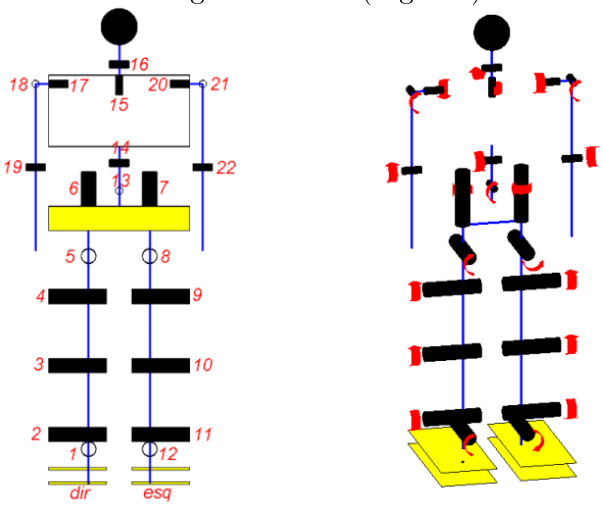


Figura 2.1: Representação 3D dos 22 graus de liberdade

A arquitectura distribuída de controlo é constituída por três tipo de unidades: Unidade de Processamento, Master e Slaves, formando uma rede de controladores interligada por um barramento CAN em configuração Master/Multi-Slave (Fig. 2.2).

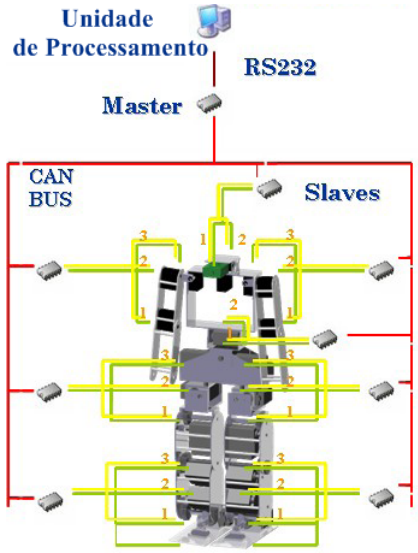


Figura 2.2: Arquitectura de Comunicações

A plataforma é constituída por um total de 10 unidades de controlo, 8 unidades Slave, uma unidade Master e a Unidade de Processamento. A comunicação entre as unidades é feita por RS-232 @115200bps, entre Master e Unidade de Processamento, enquanto que a comunicação Master/Slave é realizada por CAN (fullCAN 2.0A)3.3Kbps. É também possível, em caso de necessidade, substituir qualquer uma das unidades Slave por outra apenas tendo que mudar o endereço da mesma para o valor adequado[1]. Com esta arquitectura pretende-se que tanto o *hardware* como o *software* destas unidades seja idêntico (Fig. 2.3) de forma a obter um maior grau de fiabilidade. Assim, anomalias podem ser mais facilmente detectadas e corrigidas.

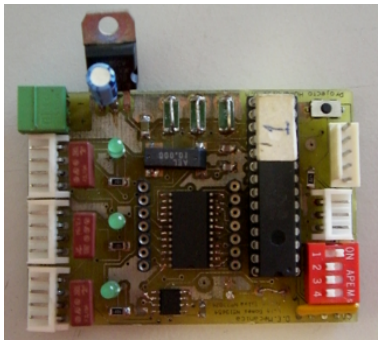


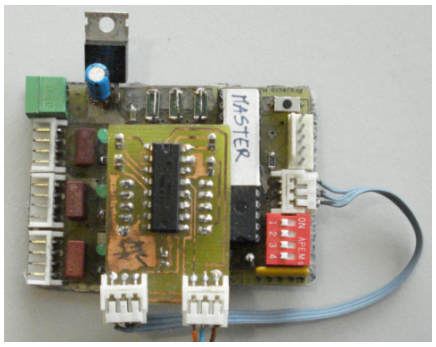
Figura 2.3: Placa genérica Master/Slave

As 8 unidades Slave (escravo) responsáveis pelo controlo local dos actuadores e sensores das juntas, encontram-se distribuídas de forma a agrupar conjuntos de três actuadores relativos a um determinado membro, como é o caso das pernas ou dos braços. As principais funções das unidades Slave são a geração do PWM de controlo dos servomotores e a aquisição dos sinais dos diversos sensores da plataforma. Cada unidade Slave é responsável por três juntas, sendo

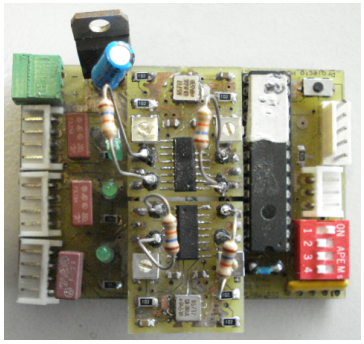
[1] Isto é feito por *hardware* através de um *DIP switch*.

que juntas com as do joelho e tornozelo (pé) estão agrupadas na mesma Slave. Uma vez que estas estão directamente relacionadas no movimento, isto permite, por aquisição dos sinais analógicos dos sensores de força instalados nos pés, controlar o equilíbrio em malha fechada. Existe assim um controlo local independente do resto do sistema sem que haja necessidade permanente de interagir a todo o momento com a Unidade Central de Processamento. Os sinais analógicos adquiridos provêm dos seguintes sensores (Fig. 2.4):

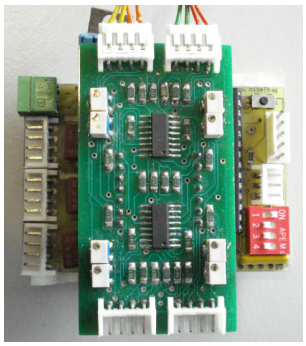
- Potenciómetros internos aos servomotores, indicativos da sua posição angular;
- Extensómetros presentes na base do pé, para medição da força de reacção;
- Acelerómetros/inclinómetros que medem a inclinação da secção da plataforma onde estão inseridos;
- Giroscópios para compensação das forças dinâmicas exercidas sobre o robô.



(a) Master com módulo de comunicação RS-232



(b) Slave com acelerómetros



(c) Slave com placa de aquisição dos extensómetros

Figura 2.4: Diversas configurações da placa genérica

Todos os valores adquiridos são registados pelos Slaves e enviados para o Master/Unidade de Processamento, que por sua vez, envia mensagens aos Slaves, que incluem:

- Posições finais que os servomotores devem atingir;
- Velocidade a que se devem deslocar;

- Tipo de controlador a ser utilizado por cada um dos servomotores ligados ao Slave;
- Parâmetros de compensação para os algoritmos de controlo;
- *Flags* booleanas de controlo (*e.g.*, PWM activo).

A unidade Master é responsável pela gestão da informação de actuação/sensorial, comunicação entre Slaves e Unidade de Processamento e ainda pelo controlo de tráfego na rede CAN.

A Unidade de Processamento assume-se como a responsável pela gestão global dos procedimentos, sendo que as suas principais tarefas passam pelo cálculo das configurações que as juntas devem adoptar com base em directivas de alto nível, processamento do sinal vídeo e actuação na plataforma com base na realimentação visual e ainda possível interacção com computador externo para monitorização, debug ou teleoperação. A Unidade de Processamento possui um ambiente de desenvolvimento versátil que permite que o desenvolvimento seja feito directamente na Unidade de Processamento, quer em linha de comandos, quer em ambiente gráfico quando necessário, ou externamente e depois compilado e executado. Esta unidade possui ainda várias interfaces de comunicação, tais como portas USB, porta Série RS-232 e Ethernet. Existe ainda a possibilidade de expansão de recursos adicionando os módulos pretendidos, uma facilidade que foi aproveitada no âmbito deste trabalho para adicionar um módulo PCMCIA. Este foi usado para permitir a ligação de uma câmara FireWire através de uma placa PCMCIA/FireWire.

O sistema de visão inclui a Unidade de Processamento, responsável pelo processamento de imagem, uma câmara FireWire suportada por uma unidade *Pan-Tilt* (PTU), sendo que grau de liberdade *tilt* da visão é complementado pelo tronco que também possui capacidade de *tilt*(Fig. 2.5).

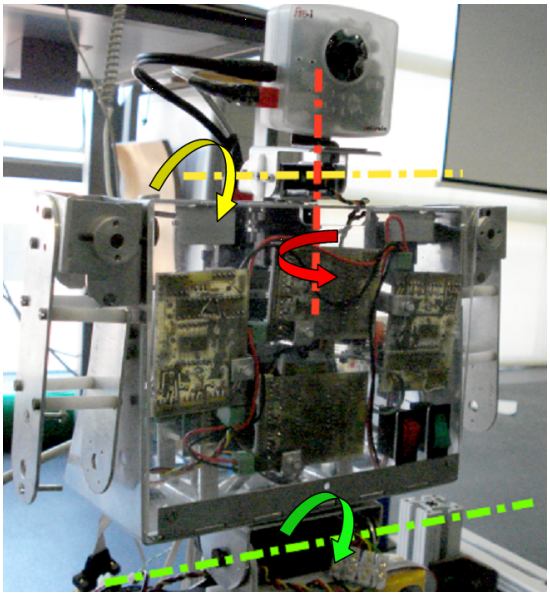


Figura 2.5: Unidade *Pan-Tilt*

Os três graus de liberdade envolvidos, nesta primeira fase, no seguimento visual encontram-se agrupados num única Slave de modo simplificar o processo de envio de comandos por parte

do Master. O controlo é feito, tal como para os outros Slaves/servomotores da arquitectura, através de um sinal de PWM gerado pelo microcontrolador do Slave. Após o cálculo dos parâmetros do movimento a efectuar, estes são enviados da Unidade de Processamento para as Slaves, que geram o sinal de PWM necessário ao movimento. Cada unidade slave, possui dois controladores em cascata:

- Controlador de primeiro nível, responsável por realizar trajectórias de vários parâmetros de actuação diferentes como, por exemplo, posição angular ou centro de pressão;
- Controlador local, responsável pela garantia da aplicação da posição angular solicitada.

Estão implementados quatro tipos de controladores de primeiro nível, o que se pode constatar na Tabela 2.1.

Tipo de controlador	**Parâmetro de actuação**	**Sensores utilizados**
Sem controlador	Posição angular do servo	Potenciómetro do servomotor
Controlo do Centro de Pressão	Centro de Pressão	4 Extensómetros
Controlo de Inclinação	Inclinação	Acelerómetros/Inclinómetros
Controlo de Velocidade Angular	Velocidade Angular	Giroscópios

Tabela 2.1: Tipos de controlo local de primeiro nível

Se nenhum controlador for seleccionado, as trajectórias efectuadas terão como base apenas posições angulares referência, relativas aos servomotores. Caso contrário, um dos conjuntos de sensores "especiais" será utilizado, calculando, o controlador, a posição angular a atribuir aos motores de modo a alcançar o valor-referência. Além da selecção do controlador, existe também a possibilidade de configurar o ganho respectivo para ajustar a sua reactividade a variações do sinal de erro.

O controlador local é responsável por garantir a aplicação da posição angular solicitada pelo controlador de primeiro nível, com base na posição medida através do potenciómetro interno ao servomotor. Caso o controlador de primeiro nível esteja desactivado, a posição de actuação enviada pela Unidade de Processamento é aplicada directamente neste controlador. Esta componente trata-se de um controlador do tipo PI que, a partir da posição referência e da medida pelo potenciómetro (sinal de erro), calcula a compensação a atribuir ao servo para que o sinal de erro se anule. Para o ajuste deste controlador é possível definir os ganhos K_I e K_P das componentes integradora e proporcional, respectivamente. Este controlo é usado para executar, por exemplo, deslocações baseadas em duração do movimento. Passando ao Slave a duração do movimento pretendida, em ciclos de 20ms, o sinal de PWM é gerado de forma a cumprir o pretendido.

2.2 Alterações à Estrutura

Para cumprir o objectivo de integrar a Unidade de Processamento na plataforma, houve necessidade de fazer adaptações pois a estrutura não possuia espaço suficiente. Mais concretamente, a cavidade do tronco que se apresentava como ponto de partida para a integração

possuía as dimensões 186×97×50(mm) (Fig. 2.6), enquanto que o conjunto Unidade de Processamento e módulo de expansão PCMCIA ocupam um volume total equivalente às dimensões 137×96×72(mm).

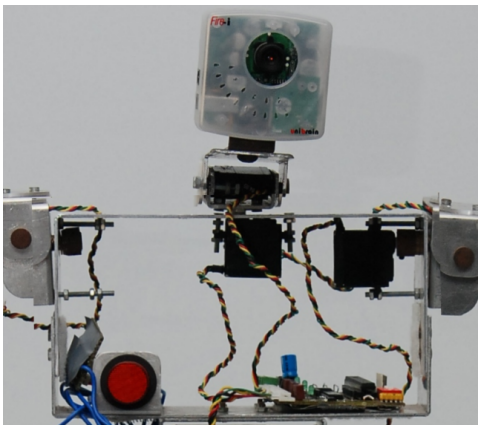


Figura 2.6: Cavidade do tronco do Humanóide

Acresce a isto, o facto de ser necessário albergar, também na cavidade do tronco, três servomotores correspondentes aos braços e PTU (*pan*) (dimensões individuais: 42×32×14(mm)), um par de baterias para suprir as necessidades energéticas (60×36×37(mm)), a unidade Master e três unidades Slaves (66×47×23(mm)) e ainda a nova placa de distribuição de alimentação (60×40×30(mm)). Verifica-se que o espaço é claramente insuficiente, sendo que a solução encontrada foi a de expandir a cavidade.

Para o efeito foi então criada uma nova peça semelhante à da Fig. 2.7 com dimensões de base 186×110(mm) e uma lateral trapezóidal com base maior e menor de 110mm e 50mm, respectivamente.

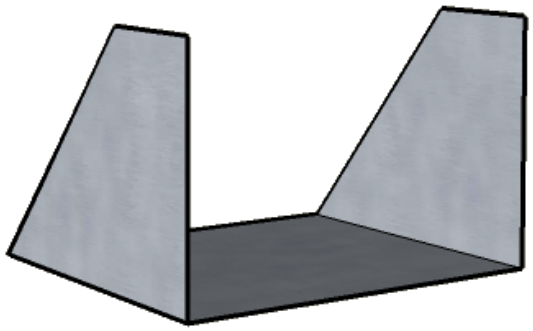


Figura 2.7: Modelo 3D do tronco actual

Ao novo *design* da cavidade foi ainda acrescentado um painel frontal (Fig. 2.8) que tem como funções, além do óbvio aumento de protecção do elementos contidos na cavidade, servir como ponto de fixação das unidades Master/Slave e ainda dos interruptores da placa de alimentação.

Este novo suporte conferiu à plataforma um acréscimo de 60mm de altura, localizados na cavicade do tronco, o que combinado com o alargamento da base e com o painel frontal

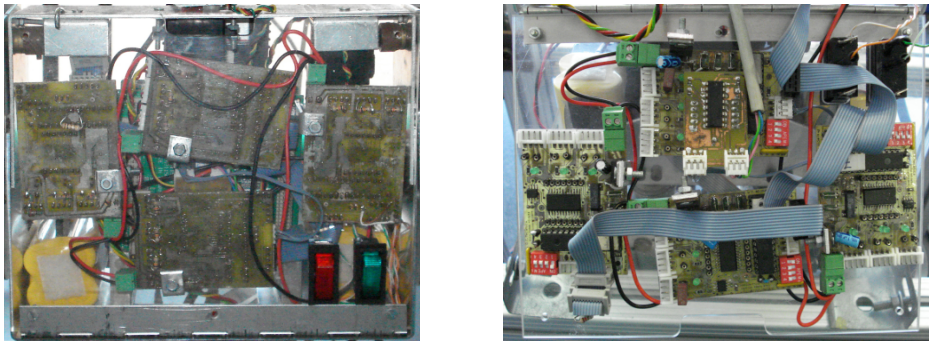


Figura 2.8: Painel frontal

permitiu a inclusão dos elementos referidos, como pode ser verificado na Fig. 2.9.

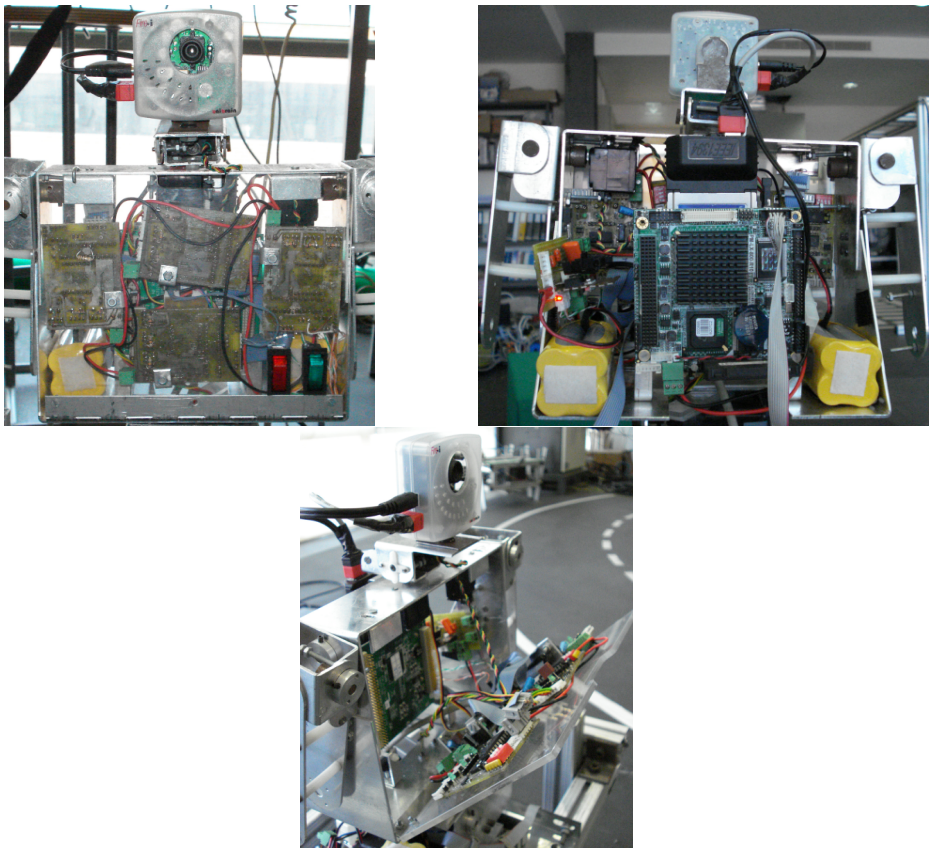


Figura 2.9: Cavidade peitoral com todos os elementos

Um outro motivo de alteração foi a placa distribuidora de alimentação. A versão existente apenas contemplava a entrada de potência de duas baterias e a saída para o circuito intermediada por um interruptor (Fig.2.10).

Com a integração da Unidade de Processamento é necessário que a placa lhe forneça alimentação bem como à câmara FireWire e ainda suporte as duas baterias adicionais (Fig. 2.11).

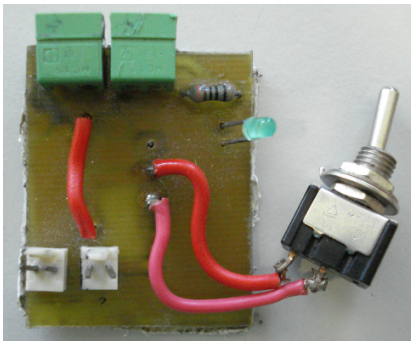


Figura 2.10: Placa distribuidora de alimentação anterior

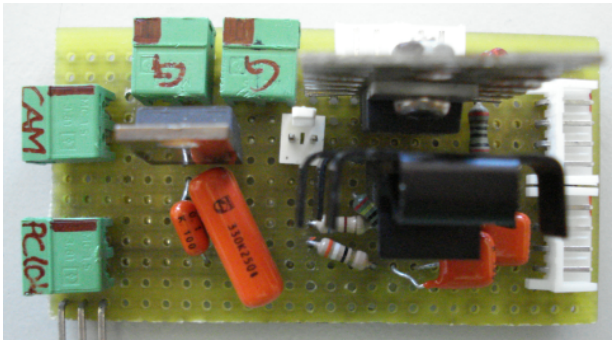


Figura 2.11: Placa distribuidora de alimentação provisória

A Unidade de Processamento consome no máximo +5V@1,13A (LX800 - 500MHz, DDR 400/1Gb RAM) e o módulo PCMCIA consome tipicamente +5V@70mA enquanto que a câmara FireWire necessita de 8-30V, sendo tipicamente alimentada a 12V e consome 1W de potência em funcionamento[14]. Como as baterias completamente carregadas têm uma tensão não inferior a 7,4V e fornecem 4800mAh, o problema que se põe é conseguir obter os 12V para a câmara. A solução implementada foi a inclusão, em série com as baterias, de uma pilha de 9V, elevando assim a tensão para ≈17V, sendo estes posteriormente regulados para 12V. A alimentação da Unidade de Processamento é efectuada regulando a tensão das baterias para 5V, sendo a alimentação do módulo de expansão feita pelo barramento PC104. A regulação é feita usando dois LM317 em paralelo de modo a suprir todas as necessidades de corrente do conjunto. Existem duas linhas de alimentação independentes no circuito, alimentação Unidade de Processamento/módulo PCMCIA e câmara FireWire, e alimentação dos servomotores que é feita directamente das baterias, sendo que ambas as linhas possuem um interruptor independente à entrada do circuito. A alimentação dos servomotores encontra-se dividida em duas saídas, uma para a parte inferior da plataforma (ancas e pernas) e outra para a parte superior (tronco, braços e cabeça)(Fig. 2.11, e Fig.2.12).

Foi ainda efectuada uma ligeira alteração na configuração da PTU (Fig. 2.13), com o objectivo de reduzir o momento provocada pelo peso da câmara e consequentemente o esforço do servomotor responsável pelo *tilt*.

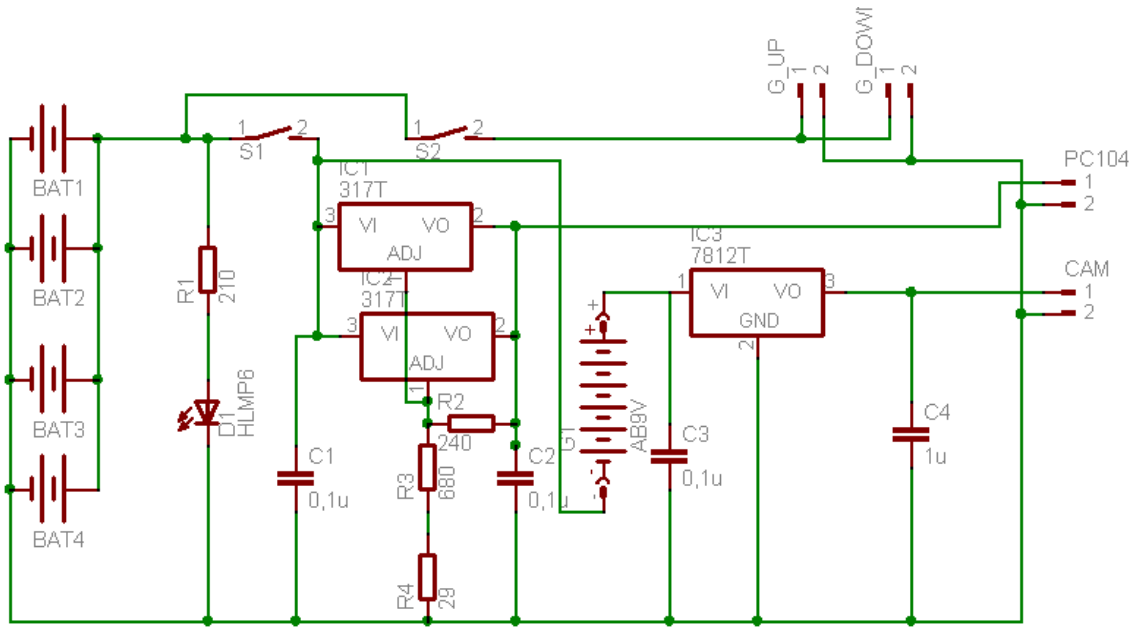


Figura 2.12: Esquema da placa distribuidora de alimentação provisória

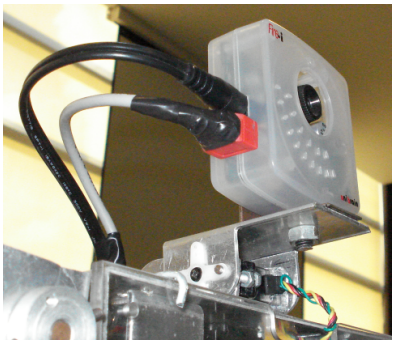


Figura 2.13: Estrutura da Unidade *Pan-Tilt* com a câmara

Capítulo 3

A Unidade Central de Processamento

3.1 Introdução

A Unidade Central de Processamento utilizada na plataforma Humanóide é um computador embebido[1] do tipo PCI-104. Esta unidade foi a escolhida em trabalhos anteriores[15], de entre outras possibilidades (entre as quais PDA's, Mainboards e outras PC/104), pela sua versatilidade e possibilidade de expansão por módulos.

Foi instalada uma versão de sistema operativo Linux mínima, o que significa que vários pacotes necessitaram de ser instalados *a posteriori*. Será abordado, neste capítulo, a instalação do sistema operativo, quais os pacotes que foram instalados, o processo de instalação e a sua respectiva configuração, quando necessária.

A Unidade de Processamento usada, como já referido, possui recursos de memória limitados, tornando assim, também esta, uma questão crítica de gestão. A utilização de um disco rígido magnético não é viável pois estes ocupam um volume demasiado grande para poderem ser usados em muitos sistemas embebidos, o nosso caso. Como é comum neste tipo de sistemas, foi utilizado uma memória compacta do tipo *Solid State Drive (SSD)*.

3.2 PC/104

3.2.1 PC/104 - Conceito

Antes de diferenciar a norma PCI-104 é interessante abordar a norma PC/104 pois esta foi a primeira existente e da qual derivam as normas posteriores, PC/104-Plus, PCI-104, PCI/104 Express, EBX e EPIC.

PC/104 (ou PC104) [16] é um standard para computadores embebidos controlado pelo *PC/104 Consortium* que define tanto a norma como o barramento do computador. O nome advém dos populares computadores pessoais inicialmente desenhados pela *IBM*, chamados PC, e do número de pinos usados para interligar as placas, 104[17]. As necessidades energéticas são reduzidas para irem de encontro às necessidades dos sistemas embebidos. Devido ao facto do PC/104 ser essencialmente um PC com uma norma diferente, a maioria das ferramentas de desenvolvimento usadas nos PCs podem ser usadas num sistema PC/104. Isto reduz o custo de aquisição de novas ferramentas e ainda a curva de aprendizagem dos programadores

[1] ou *Embedded*, é um sistema no qual o (Micro)Processador é dedicado ao dispositivo ou sistema controlado por ele.

e *hardware designers*. A norma PC/104 foi inicialmente elaborado pela *Ampro Computers* em 1987, sendo mais tarde standartizado pelo *PC/104 Consortium* em 1992. Um standard IEEE correspondente ao PC/104 foi proposto como IEEE P996, mas nunca ratificado.

Os sistemas baseados em PC/104 são usados em variados locais incluindo fábricas, laboratórios, veículos e praticamente em qualquer lugar onde dispositivos necessitam de ser controlados por um computador programável. Este é ainda direccionado para ambientes computação embebida especializada, onde aplicações dependem de aquisições de dados fiáveis, apesar de eventuais ambientes desfavoráreis. O facto de ser relativamente acessível benificia aqueles que necessitam de um sistema versátil, robusto, sem a necessidade de muito tempo de design e pesquisa.

Ao contrário da popular norma ATX[2] que utiliza o barramento PCI e é utilizado actualmente pela maioria dos PCs, a norma PC/104 não possui um *backplane*[3], o que permite que os módulos sejam interligados como blocos de construção (Fig. 3.1).

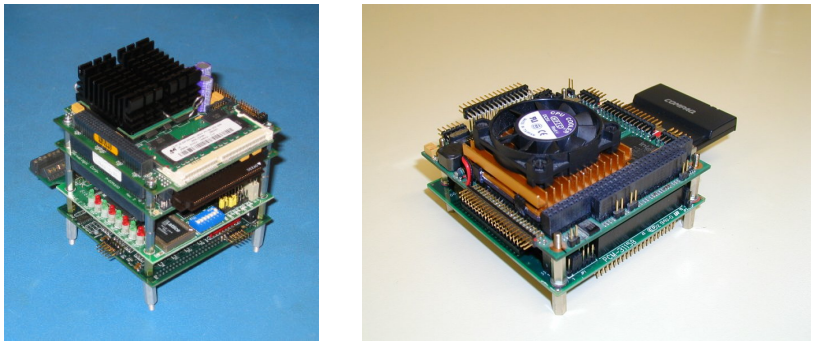


Figura 3.1: Exemplos de interligação de módulos

A ligação dos barramentos é, obviamente, mais robusta que nos PCs típicos. Isto é um resultado dos furos de montagem existentes nos cantos de cada módulo, que permitem que as placas sejam fixadas umas às outras com a ajuda de suportes isoladores. A dimensão standard das placas é 90,17×95,89 mm (3.55×3.775 polegadas), enquanto que a altura é tipicamente delimitada pelas fronteiras dos conectores. Esta delimitação de altura garante que os módulos não vão interferir com os seus vizinhos. Quase a totalidade do que se possa imaginar está disponível como módulo PC/104. Isto inclui elementos comuns como CPUs, interfaces I/O Série, portas de expansão PCMCIA, e controladores de vídeo; mas também as mais exóticas como receptores GPS, fontes de alimentação para veículos, comunicações sem fios. Uma pilha de módulos pode ser ligada como componente de uma placa de circuitos maior, ou simplesmente usada para operar *per si*.

[2]ATX - Advanced Technology Extended, norma creada pela *Intel* em 1995. ATX substitui o AT como a norma por defeito para novos sistemas devido a correções a detalhes que frustavam os constructores dos sistemas.

[3]backplane: Uma placa de circuito impresso com conectores nos quais outras placas são ligadas. Um *backplane*, tipicamente é apenas uma interligação e não possui muitos componentes activos. Isto contrasta com as *motherboards*.

3.2.2 Diferenças entre Normas

A norma PC/104 deu origem a outras normas, que se distinguem pelo tipo de barramento utilizado ou pelo tamanho. As diferenças entre normas encontram descritas de seguida e ilustradas nas Fig. 3.2 e 3.3.

- **PC/104**
  O barramento de computador PC/104 utiliza 104 pinos. Estes incluem todas as linhas normais usadas no barramento ISA[4], com pinos adicionais de massa para assegurar a integridade do barramento. Os tempos dos sinais e níveis de tensão são semelhantes ao barramento ISA mas com menores exigências de corrente.

- **PC/104-Plus**
  A norma PC/104-Plus adiciona o suporte para o barramento PCI, adicionalmente ao barramento ISA do PC/104 standard. O nome deriva da sua origem: um módulo PC/104-Plus possui um conector PC/104 (ISA) mais (em inglês, *plus*) um conector PCI-104(PCI).

- **PCI-104**
  A norma PCI-104 inclui o conector PCI , mas não o ISA, de forma a aumentar a área real da placa. O standard PCI-104 é incompatível com placas PC/104.

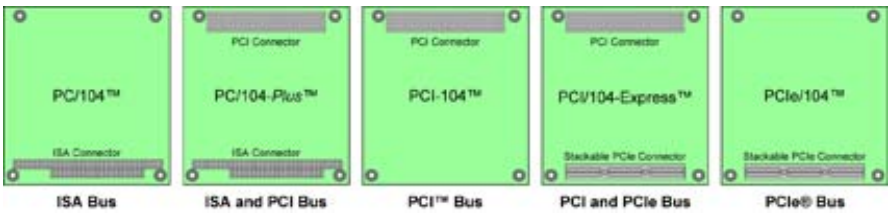


Figura 3.2: Diferenças entre Normas PC/104

- **EBX**
  EBX (*Embedded Board eXpandable*) é uma norma de computador de apenas uma placa, 146,06×203.21 mm(5.75×8.00 polegadas). O EBX é baseado no IEEE-P996 (ISA), PC/104, PC/104-Plus, PCI e PCMCIA. EBX supporta placas de expansão PC/104.

- **EPIC**
  EPIC (*Embedded Platform for Industrial Computing*) é uma norma de computador de apenas uma placa, como o EBX, supporta placas de expansão PC/104 mas é mais pequena que o EBX. Permite implementar ligações I/O tanto com pinos ou com conectores "tipo-PC". O standard fornece zonas de I/O específicas para implementar funções como Ethernet, portas Série, I/O digital e analógico, vídeo, comunicações sem fios, e várias interfaces específicas para aplicações.

- **EPIC Express**
  EPIC Express é baseado no EPIC, mas adiciona PCI Express.

[4] *ISA - Industry Standard Architecture*, um standard de barramento para computadores IBM compatíveis criado em 1981(actualmente obsoleto).

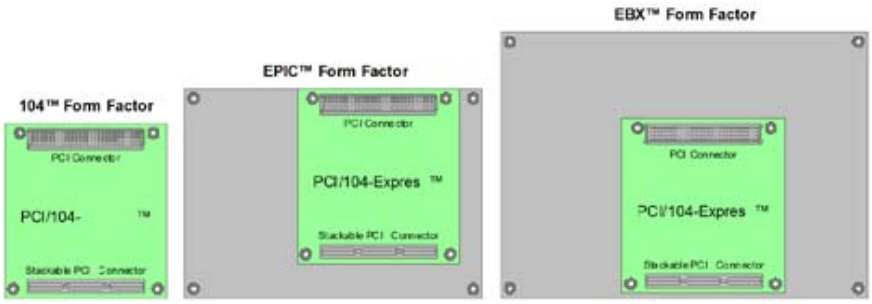


Figura 3.3: norma PC/104 vs. EBX e EPIC

3.2.3 *Stacks*

Um sistema composto de módulos PC/104, PC/104-Plus, ou PCI-104 é frequentemente referenciado como *"stack"*. Ainda que muitas *stack* incluam módulos que partilham a mesma norma, não é incomum encontrar módulos PC/104 numa *stack* que também tenha módulos PC/104-Plus.

Cada *stack* deve conter pelo menos uma *motherboard* ou CPU, que funcionará como controlador de todos os componentes periféricos. Este controlador deve suportar os barramentos de sinal usados em todos os módulos adicionados. É, contudo, possível que uma placa periférica possa desempenhar funções *per si* sem a presença de um controlador. Não existe um limite rígido para o número de placas PC/104 que podem coexistir num sistema. Contudo, à medida que se adicionam módulos, a altura da *stack* aumenta, e requisitos de sinais podem não ser mantidos.

Uma *stack* que contenha módulos PC/104-Plus deve ser controlada por uma *motherboard* PC/104-Plus. Não contando com o controlador de PC/104-Plus, o número de placas periféricas PC/104-Plus não deve exceder quatro módulos. Isto deve-se à especificação PCI, que permite quatro componentes PCI num sistema. A mesma regra aplica-se às *stacks* PCI-104.

Quando é usado o barramento PCI (módulos PC/104-Plus ou PCI-104), todos os módulos periféricos PC/104-Plus devem ser conectados consecutivamente de um dos lados do controlador devido às exigências da sinalização do barramento PCI. Um sistema PC/104-Plus ou PCI-104 pode igualmente ter módulos PC/104, que podem ser posicionados em qualquer um dos lados da Unidade de Processamento, na posição mais afastada do(s) módulo(s) PC/104-Plus (de forma a não quebrar o barramento PCI).

3.2.4 Memória de Massa

Os sistemas PC/104 geralmente necessitam de dispositivos de armazenamento também pequenos. Os dispositivos *Compact Flash* e *Solid State Disk* (SSD)[5] são bastantes mais comuns que os discos rígidos mecânicos, maiores dimensões e mais susceptíveis a falhas em ambientes agressivos.

[5] *Solid State Drive*, Dispositivo de Memória Sólida.

♦ Solid State Drive

O dispositivo de memória de massa utilizado no Humanóide é um *Solid State Drive (SSD)*. Um *SSD* [18] funciona como um disco rígido convencional, podendo substituí-lo em qualquer aplicação. Uma vez que não possui partes móveis (Fig. 3.4), um *SSD* praticamente anula os tempos de busca, latência, e outros atrasos electro-mecânicos e falhas associadas a um disco convencional.

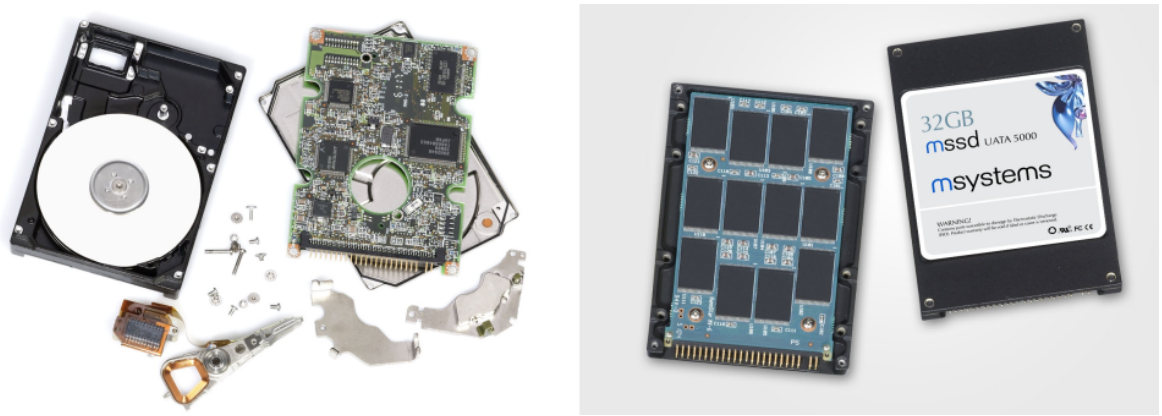


Figura 3.4: Diferenças entre o interior de um Disco Rígido e um Dispositivo de Memória Sólida

Um *SSD* é usualmente feito a partir de *flash NAND* (não-volátil) ou *SDRAM* (volátil). *SSD* baseados em memória volátil como *SDRAM* são caracterizados por acessos rápidos aos dados, menos de 0.01 milissegundos e são usados essencialmente para acelerar aplicações que seriam, de outra forma, atrasadas pela latência associada aos discos rígidos.

SSDs baseados em *DRAM* tipicamente estão associados a uma bateria interna e a sistemas de discos de *backup* para assegurar a manutenção dos dados permanentes. Se a alimentação for desligada por alguma razão, a bateria manteria o dispositivo ligado tempo suficiente para copiar todos os dados da *RAM* para o disco de *backup*. Após o restauro da alimentação, os dados são copiados novamente para a *RAM* e o *SSD* retoma o seu funcionamento normal. Contudo, existem memórias *flash* não-volátil de forma a criar alternativas mais sólidas e compactas aos *SSDs* baseados em *DRAM*, sendo este o tipo que é usado no Humanóide. Estes *SSDs* , também conhecidos como *flash drives*, não necessitam de baterias, permitindo aos replicar os tamanhos *standard* dos discos rígidos (1.8", 2.5" e 3.5").

Assim como os *SSDs DRAM*, os *SSDs flash* são extremamente rápidos. *SSDs* são especialmente úteis num computador que já tenha o máximo de *RAM* suportado. Por exemplo, algumas arquitecturas x86 possuem um limite de 4GB, isto pode ser contornado colocando o ficheiro de paginação ou o *SWAP* num *SSD*. Estes *SSDs* não garantem armazenamentos tão rápidos como as *RAM* principais devido ao estrangulamento da largura de banda que ocorre no barramento ao qual estão ligados, mas ainda assim possibilitam um aumento da performance quando comparados com a colocação do ficheiro de paginação num disco tradicional.

Comparação com disco rígidos tradicionais(*HDDs*)

Vantagens

1. Arranque mais rápido, uma vez que não é necessário por o disco a girar;

2. Mais rápidos que os discos convencionais em operações aleatórias de I/O;

3. Tempos de latência(procura) em operações de escrita e leitura muito baixos, aproximadamente 5 ordens de grandeza mais rápidos que os discos mecânicos mais rápidos.

4. Tempos de *boot* e de lançamento de aplicações mais curtos quando buscas no disco são o factor limitativo.

5. Em alguns casos, o armazenamento *Flash* tem um tempo de vida de cerca de 10 anos antes da degradação dos dados. Se os dados forem actualizados periodicamente, pode armazenar indefinidamente.

6. Poucas a nenhumas partes mecânicas.

   - Para *SSDs* pequenos, consumo de potência e produção de calor mais baixos.
   - Para *SSD*, sem ruído - A ausência de partes mecânicas torna o *SSD* completamente silencioso (apesar de alguns *SSDs* de topo incluirem ventoínhas de arrefecimento).
   - Maior fiabilidade mecânica - A ausência de partes mecânicas resulta em menores desgastes e danos. Grande capacidade de suportar "pancadas" violentas, grandes altitutes, vibrações e temperaturas, o que se aplica a computadores e outros e equipamentos portáteis, ou quando transportados.

7. Performance relativamente determinística - ao contrário dos *HDDs*, a performance dos *SSDs* é quase constante e determinística ao longo de todo o armazenamento. Isto deve-se ao facto do "tempo de busca"poder ser constante. Assim, a fragmentação tem um impacto menor na performance do que em discos mecânicos.

8. Para *SSDs* de capacidade muito pequena, menor tamanho e peso. Tamanho e peso por unidade de armazenamento ainda são melhores para *HDDs* tradicionais, mas microdrives permitem até 20GB de armazenamento numa *CompactFlash* de 42.8×36.4×5(mm).

Desvantagens

1. Preço - memórias *flash* têm preços consideravelmente mais elevados por gigabyte do que os praticados para discos convencionais.

2. Vulnerabilidade a certos tipos de efeitos, incluindo perdas abruptas de alimentação (especialmente *SSDs* baseados em *DRAM*), campos magnéticos e cargas eléctricas/estáticas comparativamente com *HDDs* convencionais (dados são armazenados dentro de uma gaiola de Faraday[6]).

3. Ciclos de escrita limitados. Armazenamento *Flash* típico desgasta-se após 100,000-300,000 ciclos de escrita, enquanto que armazemanto *Flash* de alto desempenho resiste a cerca de 1-5 milhões de ciclos de escrita (muitos ficheiros de registo, tabelas de alocação de ficheiros, e outras partes do sistema de ficheiros usadas frequentemente excedem este valor durante o tempo de vida de um computador). Sistemas de ficheiros especiais ou

[6]Gaiola de Faraday, ou escudo de Faraday é uma superfície fechada feito de material condutor, ou por uma mistura do mesmo. Esta bloqueia a passagem de campos electrostáticos externos para o interior da mesma.

definições de *firmware* podem reduzir este problema espalhando a escrita por todo o *SSD*, evitando a escrita sempre no mesmo local.

4. Escritas aleatórias lentas - como os blocos a apagar em *SSDs* são relativamente grandes, uma escrita aleatória é mais lenta que num disco convencional.

5. A vantagem *velocidade* pode ser ultrapassada pelas configurações *RAID* dos *HDDs* conventionais, que eventualmente garantem maior capacidade de armazenamento e velocidade a um preço mais baixo.

6. Em alguns casos, os *SSDs* possuem um *throughput*[7] menor que os discos convencionais. Apesar da menor latência, isto pode levar a performance dramaticamente mais baixas que os discos convencionais. *SSDs* mais caros pode ter uma largura de banda superior aos discos convencionais, por isso não é um problema universal.

♦ **Compact Flash**

É de referir que existe uma forma de expandir a memória de massa da Unidade Central de Processamento sem substituir o *SSD*, esta é adicionar uma *Compact Flash* (Fig.3.5) ao sistema. Para o efeito está disponível um *slot* na placa mãe PCI-104.

Figura 3.5: CompactFlash

3.3 Solução baseada em PC/104

A Unidade Central de Processamento adquirida é um computador embebido *PM-LX-800-R10* [19] da *iEi Technology Corp* (Fig. 3.6), standard PCI-104, com a seguinte configuração: AMD Geode LX800@500MHz, 512Mb RAM, SSD 1Gb. A totalidade das características encontra-se descritas na Tabela. 3.1.

De forma a completar as necessidades da plataforma foi ainda adquirido o módulo PCM-3115, um módulo de expansão PC/104-Plus para interface PCMCIA, de forma a conectar um adaptador PCMCIA/FireWire que servirá para comunicar com a câmara de vídeo (Fig. 3.7). A Tabela 3.2 resume as características do módulo.

[7] *throughput*, relação entre a saída de um sistema e a sua entrada; capacidade de processamento num determinado período de tempo

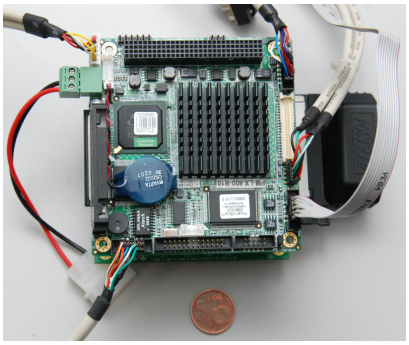


Figura 3.6: Unidade de Processamento

PM-LX-800 R10	
CPUs Supportados	AMD® Geode LX-800 @ 500 MHz
Memória Cache	64K I/ 64k D L1 cache, 128K L2 cache
Chipset do Sistema	AMD® CS5536
Controlador I/O	AMD® CS5536
Memória	Uma DDR 333/400MHz SO-DIMM SDRAM 200-pinos com um máximo de 1GB.
Interface Barramento PCI	Revisão 2.2
Super IO	W83627EHG
Display	CRT integrado no AMD® Geode LX800
TTL/ LVDS	24 bit TTL integrado no AMD LX800 18 bit LVDS
Interface HDD	suporte canais IDE
Suporte Potência	suporte AT/ATX
Consumo Potência	+5V @ 1.13A (DDR400 1GB)
Watchdog Timer	Programável por Software, suporta resets de sistema 1∼ 3255 sec.
Interfaces I/O	2 × USB 2.0 1 × LPT 1 × CFII 1 × IDE
Ethernet	10/100Base-T RTL8100C
BIOS	AWARD
Dimensões Físicas	91mm × 95mm
Peso	GW:0.65Kg; NW: 0.25Kg
Temperatura de Funcionamento	Mínima: 0°C (32°F) Máxima: 60°C (140°F)

Tabela 3.1: Especificações Técnicas da PCI-104 PM-LX-800 R10

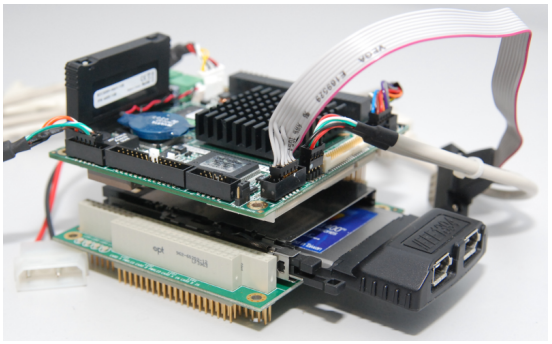


Figura 3.7: Unidade de Processamento e módulo de expansão PCMCIA

PCM-3115	
Interface Barramento	PC/104-Plus (ISA/PCI)
Chipset do Sistema	Ricoh® CardBus Controller
Modo de Compatibilidade 16-bit	Sim suporte(3E0/3E2 I/O)
Suporte GPIO	Sim.
Compatibilidade DMA PCI	Sim suportado
Barramento de dados	Barramento de dados 32-bit (Cumpre Especificações Barramento PCI 2.1)
Interface PCI	Interface 3,3V (tolera até 5V)
Configuração de Hardware	Detecção & configuração automática Plug & Play
Driver de Software	Suporta Windows NT 4.0, Windows 95/98/2000
Dimensões Físicas	90mm × 96mm
Peso	0.094 Kg
Temperatura de Funcionamento	Operação: 0∼60°C (32∼140°F) Armazenamento: -40∼85°C (-40∼185°F)
Humidade de Funcionamento	0∼90% de humidade relativa, sem condensação
Tensão de Alimentação	+5V±5% (DDR333 256MB)
Consumo Potência	+5V @ 0.07A

Tabela 3.2: Especificações Técnicas do módulo PCMCIA PCM-3115

3.4 Instalação do Sistema Operativo

3.4.1 Versão Instalada

Uma das tarefas iniciais consistiu na escolha adequada do sistema operativo (SO) a instalar na Unidade de Processamento tendo como ideais base a própria versatilidade do SO, a possibilidade de personalização, ambiente de desenvolvimento simples, baixo custo e acima

de tudo, estabilidade. Dentro destes pressupostos foi tomada a decisão de avançar para uma distribuiçao Linux, SO bastante popular e versátil. A versão instalada seria uma versão minimalista, devido principalmente à capacidade da memória de massa disponível. A versão instalado corresponde à distribuição Debian GNU/Linux 40r0 i386 Net Install[8]. Recorreu-se a esta versão por ser minimalista ($\approx$ 200Mb) e possuir, como qualquer outra distribuição Linux, as funcionalidades que permitem a sua actualização e instalação de pacotes adicionais pela rede.

3.4.2 Instalação via USB

O Humanóide é um sistema embebido, controlado por um PCI-104, um computador com características diferentes de PC normal. Devido a limitações impostas pelo sistema em uso, não possui leitor de CD/DVD, determinou-se como melhor opção, a instalação do Linux através de um dispositivo de armazenamento USB (*Pen Drive*, HD externo, etc.).

O processo de instalação está esquematizado na Fig. 3.8. Um tutorial da instalação está disponível no Apêndice A.

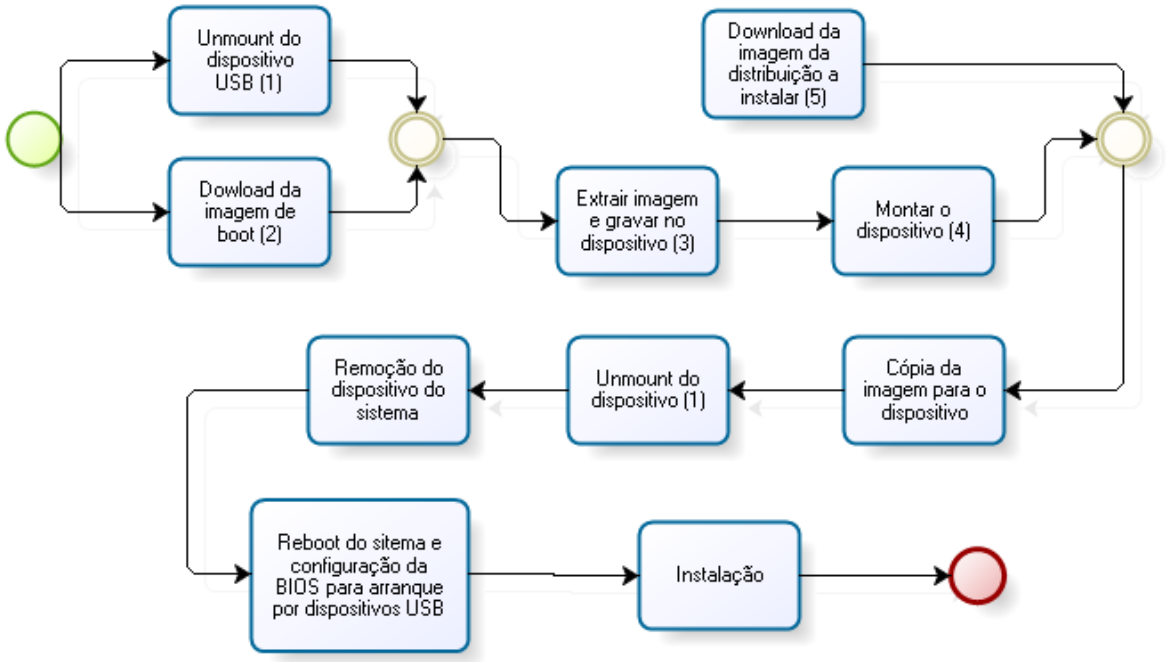


Figura 3.8: Processo de instalação do Sistema Operativo

Comandos da shell:

1. **sudo umount /dev/sda**
2. **wget** ftp://ftp.debian.org/debian/dists/stable/main/installer-i386/current/images/hd-media/boot.img.gz
3. **sudo zcat boot.img.gz > /dev/sda**

[8]Mais informaçao em http://www.debian.org/releases/stable/i386/

4. **sudo mount /dev/sda /mnt**

5. **sudo wget** http://cdimage.debian.org/debian-cd/4.0_r0/i386/iso-cd/debian-40r0-i386-netinst.iso

Nota:*Toda a instalação do Linux efectuada no Humanóide foi feita segundo os parâmetros "standard" exceptuando o Particionamento do Disco. Aqui não foi criada área de SWAP, sendo a totalidade do disco reservada para o sistema de ficheiros (ext3). A razão para esta opção será abordada na secção 3.7.*

3.5 Configuração de Rede

Como qualquer outra instalação Linux, todos os *updates*, *upgrades* e instalação de novas aplicações são tipicamente efectuadas pela Internet, sendo necessário configurar o acesso à rede. Um diagrama relativo à configuração encontra-se na Fig. 3.9. Mais detalhes da instalação encontram-se no Apêndice B.

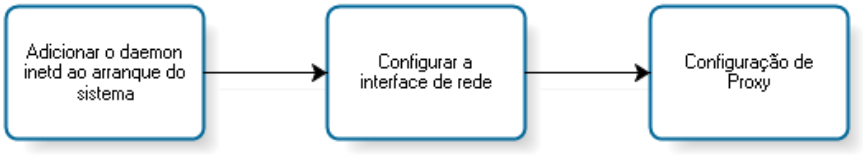


Figura 3.9: Processo de configuração de rede

3.6 Ambiente de Desenvolvimento

Para a realização deste trabalho foi necessário dotar o SO de um ambiente de desenvolvimento capaz de suprir as necessidades do utilizador/programador. Como já foi referido, a versão do SO instalada possui de raíz um conjunto mínimo de ferramentas. Assim, os pacotes de aplicações necessários, por exemplo, para o desenvolvimento do software do Humanóide (editor de código fonte, compilador, etc.) necessitaram de ser instalados pelo utilizador.

Os pormenores relativos à instalação e configuração de cada pacote encontra-se no Apêndice C.

3.6.1 Ferramentas de Sistema

♦ **Sudo**

O **sudo** é um comando do sistema operacional Unix permite a utilizadores ditos comuns obterem privilégios de outro utilizador, tipicamente o super utilizador, para executar tarefas específicas dentro do sistema de maneira segura e controlável pelo administrador. O nome é uma forma abreviada de se referir a *substitute user do* (fazer substituindo utilizador) ou *super user do* (fazer como super utilizador).

♦ **Xfce**

Xfce é um ambiente de trabalho gráfico para sistemas Unix e similares. O objectivo do Xfce é ser rápido, consumir poucos recursos de processamento enquanto se mantém visualmente apelativo e simples de usar. Este incorpora a filosofia tradicional de reutilização e modularidade do UNIX. Consiste num número de componentes que juntos fornecem a funcionalidade completa para o ambiente de *desktop*. Os módulos são empacotados separadamente e pode seleccionar os pacotes disponíveis para criar o melhor ambiente de trabalho personalizado. Outra prioridade do Xfce é a conformidade com os padrões e especificações definidos no freedesktop.org. O Xfce pode ser instalado em diversas plataformas UNIX, pode ser compilado em Linux, NetBSD, FreeBSD, Solaris, Cygwin e MacOS X, em x86, PPC, Sparc, Alpha, etc.

♦ **Coriander**

O **Coriander** é a interface gráfica para Linux que permite ao utilizador controlar uma câmara digital através do barramento IEEE1394 (também conhecido como Firewire, ou iLink). O Coriander possui várias funções e além de alterar os parâmetros de camâra, permite também tirar fotos, gravar vídeos, enviar imagens para um site FTP, converter o vídeo para um *stream* V4L[9], etc. É, obviamente, disponibilizado também imagem em tempo-real da câmara. É ainda de salientar que o Coriander funciona com qualquer câmara compatível com as especificações IIDC (também conhecidas como especificações DCAM). Estas incluem muitas das *webcams* 1394 e também a maioria das câmaras industriais e científicas. Contudo, quaisquer câmaras onde se pode inserir uma videocassete (camcorders,...) não são suportadas. Estas, gravam vídeo DV comprimido na cassete, enquanto que o IIDC especifica fluxos instantâneos de vídeo, não comprimidos.

3.6.2 Ferramentas de Edição e Compilação de Código-Fonte

♦ **Editor - Vim**

Para edição de ficheiros foi instalada a aplicação **Vim** ou *Vi Improved* que é uma versão mais poderosa e mais exigente em termos de espaço em disco e requisitos de memória do editor de texto *vi*.

♦ **Compilador - GCC**

O **GCC** ou *GNU Compiler Collection* é um conjunto de compiladores de linguagens de programação produzido pelo projecto GNU. É software livre distribuído pela Free Software Foundation (FSF) sob os termos da GNU GPL, e é um componente-chave do conjunto de ferramentas GNU. É o compilador padrão para sistemas operativos UNIX e Linux e certos sistemas operativos derivados tais como o Mac OS X. Este era originalmente designado por *GNU C Compiler* (Compilador C GNU), por apenas suportar a linguagem C, mas foi mais tarde alterado para suportar mais linguagens. A versão *standard* 4.2 do compilador suporta: C, C++ (G++), Java (GCJ), Ada (GNAT), Objective-C, Objective-C++, e Fortran (GFortran).

[9] *Streams* V4L (*Video For Linux*) são fluxos de dados utilizados em APIs (Interfaces para Programação de Aplicações) V4L.

♦ **Gestão de Compilação - Make**

O **Make** é uma aplicação concebida para compilar automaticamente o código fonte de um programa segundo as instruções previamente definidas num ficheiro chamado "Makefile" sendo capaz de resolver as dependências do programa que se pretende compilar.

♦ **Ambiente Integrado de Desenvolvimento - KDevelop**

Foi também instalado o IDE[10] **KDevelop** para uma melhor gestão dos projectos de software a desenvolver. No entanto este exige maiores recursos computacionais e uma interface gráfica (instalação da interface gráfica discutida mais atrás (Xfce)).

3.6.3 Bibliotecas de Programação

♦ **OpenCV**

O **OpenCV** (Open Source Computer Vision Library) [27] é uma biblioteca multiplataforma, originalmente desenvolvida pela Intel, para uso, incorporação e modificação por parte de equipas de pesquisa, desenvolvimento de *software* comercial, fabricantes de câmaras e governos, como se encontra descrito na sua Licença.

A filosofia subjacente é a ajuda ao desenvolvimento de aplicações comerciais da visão artificial nos interfaces humano-computador, robótica, monitorização, biométrica e segurança, fornecendo uma infraestrutura gratuita e livre onde esforços distribuídos da comunidade de visão podem ser consolidados e a sua performance optimizada. Existe uma comunidade *online* organizada em torno do OpenCV, para partilha de informação, *bugs*, opiniões, perguntas e respostas acerca da biblioteca. A documentação relativa, *OpenCV Wiki-pages*, encontra-se agora em páginas possíveis de serem editadas pelos utilizadores[11].

O foco é o processamento de imagem em tempo-real, incluindo as seguintes áreas de aplicação:

- Interface Humano-Computador (IHC);
- Identificação de objetos;
- Sistema de reconhecimento facial;
- Reconhecimento de movimentos;
- Gravação de vídeos;
- Robôs móveis.

3.7 Gestão de Memória

Um dos grandes inconvenientes do dispositivo de memória usado, *Solid State Drive*, é o seu limitado número de ciclos de escrita. Estes dispositivos, ao fim de cerca de 100,000-300,000

[10] IDE - *Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento é uma aplicação que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

[11] http://opencvlibrary.sourceforge.net

ciclos de escrita deterioram-se e começam a apresentar sectores danificados, o que põe em causa a segurança da informação armazenada.

Os principais responsáveis por acessos a disco num sistema operativo são as tarefas de sistema relacionadas com a área de *Swap* e *Journaling* dos sistemas de ficheiro *ext3* e ainda o utilizador. Sempre que um ficheiro é criado, alterado ou apagado, existem vários acessos ao disco.

O objectivo desta secção é analisar os pontos referidos e apresentar soluções compatíveis com o nosso sistema.

3.7.1 Área de *Swap*

Devido às limitações, ciclos de escrita limitados, discutido na secção 3.2, do dispositivo de memória de massa usado, um *Solid State Drive*, decidiu-se prescindir da área de *Swap* e assim reduzir uma das fonte de acessos ao disco mais frequente.

A área de *Swap* é uma zona de memória reservada no dispositivo de memória de massa na qual o SO escreve como complemento à memória *RAM*. Esta é usada para descarregar da *RAM* informação de programas que se encontram inactivos, libertando assim, espaço para novos programas ou programas que tinham sido descarregados para a área de *Swap* e voltaram a estar activos.

Isto foi alcançado eliminando a partição dedicada à área de *Swap* no processo de instalação do SO.

3.7.2 Journaling

O sistema de ficheiros padrão do Linux é o *ext3*, *Third Extended File System*, um sistema de ficheiros com *journaling* que está disponível no *kernel* desde a versão 2.4.15. O *journaling* é uma técnica utilizada por vários sistemas de ficheiros, que proporciona uma recuperação rápida de problemas, pois não necessita de uma verificação completa de todo o sistema de ficheiros. Ao invés, é mantido um *journal* que regista cada abertura de ficheiro, permitindo saber que ficheiros poderão ter sido afectados no caso do sistema ter sido mal desligado ou por uma perda de alimentação. Todas as alterações são registadas no *journal* (tipicamente um registo circular numa zona especialmente alocada da partição) antes de serem efectivamente escritas no disco.

Motivação

Os sistemas de ficheiros tendem a ser estruturas de dados muitos grandes (complexas); actualizá-las para reflectir alterações em ficheiros e directorias normalmente requer várias operações de escrita independentes. Isto introduz uma condição de corrida[12], na qual uma interrupção (como uma falha de alimentação ou um bloqueio de sistema) pode deixar estruturas de dados num estado intermédio inválido. Por exemplo, apagar um ficheiro num sistema de ficheiros Unix envolve dois passos:

1. Remover a sua entrada no directório.

[12]Uma condição de corrida é uma falha num sistema ou processo pela qual a saída e/ou o resultado do processo se torna inesperada e criticamente dependente da sequência ou sincronismo de outros eventos. O termo tem origem na ideia de dois sinais disputarem uma corrida para influenciar a saída primeiro.

2. Marcar o espaço ocupado pelo ficheiro e o seu *inode*[13] como livres no mapa de alocação.

Se o passo 1 ocorrer imediatamente antes de um bloqueio de sistema, vai passar a haver um *inode órfão*, contém informação de um arquivo que já não consta da estrutura de dados, e consequentemente uma perda de capacidade de armazenamento. Por outro lado, se apenas o passo 2 é executado antes do bloqueio, o ficheiro ainda não apagado vai ser marcado como livre e possivelmente ser escrito por cima.

A recuperação num sistema de ficheiros sem *journaling* requer uma análise completa de toda a estrutura de dados para encontrar e corrigir quaisquer inconsistências. Este processo pode ser relativamente lento se o sistema de ficheiros for grande, e existir pouca largura de banda I/O disponível. Um sistema de ficheiros *journaled* mantém um registo de todas as alterações que se pretendem fazer, antes mesmo destas serem efectuadas. Após um bloqueio, o processo de recuperação consiste na recriação de todas as alterações até que o sistema de ficheiros volte a ser consistente. As alterações são assim chamadas de *atómicas* (ou indivisíveis) pois apenas podem surgir duas opções:

- Sucesso (tiveram sucesso originalmente ou foram recriadas completamente durante a recuperação), ou
- Não são recriadas de todo.

Tipos de *Journaling*

O *journaling* pode ter um forte impacto no desempenho porque requer que toda a informação seja escrita duas vezes. *Journaling* baseado em Metadados[14] é um compromisso entre fiabilidade e desempenho que apenas regista no *journal* alterações aos metadados dos ficheiros. Isto continua a assegurar uma recuperação rápida do sistema de ficheiros no próximo *mount*, mas abre uma janela para a corrupção dos dados pois dados não regitados e metadados registados podem estar assíncronos.

Por exemplo, adicionar dados a um ficheiro num sistema de ficheiros Unix requer três passos:

1. Aumentar o tamanho do ficheiro no seu *inode*.
2. Alocar espaço para os dados adicionais no mapa de alocação.
3. Escrever os dados no espaço recém alocado.

Num *journal* baseado em metadados, o passo 3 não seria registado. No caso deste não ser efectuado, mas os passos 1 e 2 serem recriados durante a recuperação, o ficheiro teria lixo adicionado em vez dos dados originais. A *cache* de escrita na maioria dos sistemas operativos ordena a sua escrita para maximizar o *throughput*. Para evitar o perigo de escritas desordenadas com este tipo de *journaling*, o ordenamento da escrita tem de ser tal que garanta que os dados são escrito para o disco antes dos seus metadados. Na implementação do ext3 em Linux existem três tipos de *journaling* disponíveis:

[13] *inode*, nó de indexação, é uma estrutura de dados constituinte de um sistema de ficheiros que segue a semântica Unix. O *inode* armazena informações sobre arquivos, tais como dono, permissões e a localização do mesmo (*e.g.*, blocos alocados num volume de disco).

[14] Metadados são dados capazes de descrever outros dados, ou seja, dizer do que se tratam, dar um significado real e plausível a um ficheiro de dados, são a representação de um objecto digital.

- ***Journal***
  (lento, mas mais seguro) Tanto os metadados como o conteúdo do ficheiro são escrito no *journal* antes de serem introduzidos no sistemas de ficheiros. Isto optimiza a consistência com uma perda de performance porque todos os dados são escritos duas vezes. Sem esta configuração em **/etc/fstab**, um ficheiro a ser editado *in loco* arrisca-se a ficar corrompido se ocorrer uma falha de energia ou um *kernel panic*, dependendo da forma como a aplicação está a escrever no ficheiro.

- ***Ordered***
  (velocidade média, risco médio) *Ordered* funciona como *Writeback* mas força o conteúdo do ficheiro a ser escrito antes de os seus metadados serem registados no *journal*. Esta é a configuração por defeito em muitas distribuições Linux.

- ***Writeback***
  (a mais rápida, mais arriscada; de certa forma equivalente ao ext2[15]) Os metadados são registados no *journal* mas os dados do ficheiro não. Isto torna o processo mais rápido, mas introduz o risco acontecerem escritas fora de ordem, por exemplo, se ocorrer um bloqueio durante o processo de acrescentar informação a ficheiros, estes podem passar a ter lixo associado no próximo *mount* do sistema de ficheiros.

3.7.3 RAM Disk

Uma *RAM disk* é uma porção de *RAM* que é usada como se fosse um disco normal. As *RAM disk* têm tamanhos fixos e funcionam como uma partição normal. Os tempos de acesso são mais baixos do que os de um disco rígido normal. Contudo, dados armazenados numa *RAM disk* são perdidos quando o sistema é desligado ou a energia é desligada. As *RAM disk* são um bom lugar para armazenar e trabalhar com dados temporários. O Linux possui suporte *built-in* para *RAM disks* apartir da versão 2.4 do *kernel*. *RAM disks* são criadas para diversos fins, entre os quais:

- Trabalhar com dados desencriptados de documentos encriptados;
- Servir alguns tipos de conteúdos web;
- Montar sistemas de ficheiros *Loopback* (como distribuições run-from-floppy/CD)[16].

A utilização de *RAM disks* revela-se uma boa solução para minimizar o número de acessos ao disco durante a utilização do sistema para, por exemplo, desenvolvimento de código. Todos as alterações e criações de ficheiros são realizadas na *RAM* durante o desenvolvimento. No entanto, no final é necessário proceder à gravação de todo os ficheiros criados/editados no suporte físico, ou a informação será perdida.

Um tutorial de criação de uma *RAM disk* encontra-se no Apêndice D.

[15] ext2, ou *Second Extended File System* é um sistema de ficheiros para o *kernel* do Linux. É o antecessor do ext3 e ainda não possuía *journaling*.

[16] exemplo: as distribuições Live CD de Linux

Capítulo 4

O Sistema de Visão

4.1 Introdução

Um dos principais objectivos deste trabalho é desenvolver o sistema de visão que irá permitir ao Humanóide a interacção com o meio envolvente. O desenvolvimento de um sistema eficiente representa um desafio, considerando que esta é uma plataforma de pequenas dimensões, que se pretende autónoma. Ao longo dos últimos anos, tem sido realizada uma intensa investigação no sentido de dotar os robôs humanóides futebolistas com capacidades de visão [21, 39]. Uma abordagem comum tem sido a utilização de visão omnidireccional para a detecção de objectos [20, 21, 22]. No entanto, esta abordagem não reflecte uma aproximação ao modelo biológico. O desenvolvimento de arquitecturas de visão para robôs humanóides deverá passar mesmo pela solução de inspiração biológica, encontrando-se já restrições no regulamento da competição *RoboCup* que apontam para isso mesmo [63].

A base deste projecto consiste na utilização de uma arquitectura de controlo distribuída, para fornecer capacidade de processamento suficiente para a implementação de comportamentos baseados em estímulos externos [23]. Assim, a visão monocular activa capaz de localizar, de se alinhar com e seguir um objecto de interesse torna-se uma primeira aproximação ao seu equivalente biológico [61, 62].

Uma vez que um dos objectivos deste projecto é a participação na competição internacional Robocup, na categoria de futebol robótico, é necessário que o Humanóide consiga identificar os elementos que o rodeiam no campo de jogo. Entre esses elementos encontram-se as linhas delimitadoras do campo, as balizas, os adversários e um elemento fundamental que é uma bola. Assim, o passo inicial consistirá na identificação do objecto, a bola, e na determinação da sua localização no plano de imagem e posição relativa ao robô, e de seguida a actuação na plataforma para que esta possa fazer o seguimento da bola recorrendo à *pan-tilt* (PTU). Como a câmara usada possui uma lente com pequeno campo de visão, o grau de liberdade do tronco, no plano sagital, é usado para alargar o campo de visão, permitindo ao robô ver os seus próprios pés.

Neste capítulo são introduzidos alguns fundamentos da visão por computador e é apresentada a arquitectura do sistema de visão. São, ainda, abordados os problemas do seguimento de um objecto usando uma única câmara integrada numa unidade PTU. É descrito todo o processo de percepção-decisão-actuação, incluindo as questões de aquisição, processamento de imagem e do ciclo de controlo.

4.2 Fundamentos de Visão por Computador

Visão por computador é a ciência que desenvolve as bases teóricas e algorítmicas pelas quais a informação acerca do mundo pode ser automaticamente extraída e analisada a partir de uma imagem observada, de um conjunto de imagens, ou de uma sequência de imagens [35]. Esta informação pode ser relacionada com o reconhecimento de um objecto genérico, a descrição tridimensional de um objecto desconhecido, a posição e orientação do objecto observado, ou a medida de uma qualquer propriedade espacial do objecto, como a distância entre dois pontos distintos ou o diâmetro de uma secção circular.

Entre os tópicos abordados, a questão da robótica guiada por visão tem sido uma das áreas de investigação mais activas, iniciada há mais de três décadas. Os recentes avanços tecnológicos, nomeadamente ao nível do poder computacional, resultaram num grande número de sistemas de sucesso usando componentes comerciais (*"off-the-shelf"*). As aplicações tecnológicas variam desde a montagem robótica assistida por visão, a inspecção de medidas, a determinação de existência de defeitos, as casas inteligentes e a indústria automóvel.

Esta secção pretende apresentar uma breve introdução dos componentes e problemas associados à visão por computador, e alguns conceitos teóricos que possibilitam a implementação do sistema de um controlo com realimentação visual.

4.2.1 Componentes de um Sistema de Visão

A visão por computador é um tema muito complexo, de modo que é útil dividir o tema nas suas várias componentes. Uma maneira simples de tentar entender é estabelecer uma plataforma de comparação com um sistema biológico. Assim, as funcionalidades básicas comuns aos sistemas de visão biológicos e artificiais incluem as seguintes [36]:

- **Fonte de Radiação.** Se não houver qualquer radiação a ser emitida da cena ou objecto de interesse, nada pode ser observado ou processado. É necessário existir uma iluminação apropriada para os objectos que não emitem radiação por si.

- **Câmara.** A câmara é responsável por recolher a radiação emitida pelo objecto, de forma a que a origem da mesma possa ser referenciada à sua origem. Nos casos mais simples isto é feito apenas por uma lente. Mas em alguns casos ser algo completamente diferente, por exemplo, um espectrómetro óptico, um tomógrafo raio-x, etc.

- **Sensor.** O sensor converte a radiação recebida num sinal apropriado a posterior processamento. Para um sistema de imagem, normalmente é necessário um arranjo 2D de sensores para capturar a distribuição espacial da radiação. Com o sistema apropriado de varrimento, um único sensor ou uma fila de sensores pode ser suficiente.

- **Unidade de Processamento.** Aqui é processada a informação recolhida, extraindo características adequadas que possam ser usadas para quantificar as propriedades do objecto e categorizá-las em classes. Outro componente importante é um sistema de memória para recolher e armazenar a informação da cena, incluindo mecanismos para apagar dados desnecessários.

- **Actuadores.** Os actuadores reagem ao que resulta da observação visual. Tornam-se parte integrante do sistema de visão quando o sistema de visão responde activamente à

observação como, por exemplo, no seguimento de um objecto de interesse ou em navegação guiada por visão (percepção visual activa).

4.2.2 Áreas e Níveis da Visão por Computador

A Visão por Computador pode ser subdividida em três diferentes níveis: um baixo nível que inclui as áreas de *Percepção* e *Pré-processamento*; um nível médio que inclui *Segmentação* e *Descrição de propriedades* e um alto nível onde se encontram o *Reconhecimento* e a *Interpretação*.

O pré-processamento é aplicado aos dados da imagem para assegurar que estes satisfazem certas suposições implícitas ao método de visão utilizado. Exemplos são a redução de ruído, realce de contrastes e reamostragem da imagem para correcção do seu sistema de coordenadas. Segmentação de imagem é o particionamento de uma imagem num conjunto de regiões não sobrepostas, cuja união é a imagem completa [35]. O objectivo da segmentação é decompor a imagem em partes que são relevantes, relativamente a uma aplicação particular, simplificando e/ou mudando a sua representação para facilitar a análise [59]. A segmentação é tipicamente usada para localizar objectos e formas (linhas, curvas, etc.) em imagens. É importante notar que a precisão da fase de segmentação pode determinar em grande escala o sucesso ou falha dos procedimentos de análise de imagem por computador.

Alguns dos problemas inerentes à visão por computador ocorrem porque as unidades de observação não são as unidades de análise. A unidade da imagem digital observada é o *pixel*. O *pixel* tem propriedades de posição e valor. Por si, conhecimento da posição e do valor de um *pixel* específico tipicamente não transmite qualquer informação relacionada com o reconhecimento do objecto, a descrição da forma de um objecto, a sua posição ou orientação, a medida de uma distância no objecto, ou se o objecto possui defeitos.

O tipo de objecto, a iluminação, o plano de fundo, o tipo de sensor de imagem e o ponto de vista do sensor determinam a dificuldade do problema de visão por computador; isto é, a transformação da unidade do *pixel* para a unidade do objecto com posição e orientação pode ser directa ou complexa. Este processo de transformação pode ser difícil de concretizar. Uma enorme variedade de objectos complexos pode ser capturada devido a vários factores. Por exemplo: alguns objectos podem estar parcialmente ocultados por outros, podem ocorrer sombras, as reflexões de luz no objecto podem ser variadas e o fundo pode ser fonte de interferência.

4.2.3 Controlo com Realimentação Visual

Como qualquer outro sistema, também os sistemas de visão podem ser alvo de controlo em malha aberta ou em malha fechada. Um sistema em malha aberta utiliza as capacidades de visão para extrair dados que são usados para gerar uma sequência de controlo. No entanto, não existe interacção "on-line" entre robô e ambiente, sendo o sistema incapaz de reagir a alterações no ambiente de controlo. Um sistema em malha fechada utiliza continuamente a informação visual recolhida para reajustar a sequência de controlo ao ambiente.

O controlo visual é usado numa grande variedade de aplicações, sendo a mais comum a manipulação de objectos. Esta aplicação tipicamente requer a detecção de objectos, segmentação, reconhecimento, controlo, alinhamento e preensão. Embora muitas abordagens ao tema não contenham todos os aspectos mencionados, a tarefa de manipulação traduz uma plataforma de trabalho global a tomar em consideração nas diversas pesquisas. Existe na literatura uma

excelente introdução ao controlo visual na forma de um tutorial [40]. É usualmente feita a seguinte distinção entre as duas diferentes formas de utilizar a informação visual num sistema robótico [43, 45]:

- **Controlo do Robô em Malha Aberta**

  A extracção de informação da imagem e o controlo do robô são duas tarefas independentes onde é primeiro efectuado o processamento de imagem e de seguida gerada uma sequência de controlo. Para mover o robô baseado na informação visual extraída da imagem, a(s) câmara(s) necessita(m) de estar calibrada(s) relativamente ao robô. Adicionalmente, os modelos cinemáticos directo e inverso são necessários para a conversão de posições do robô no espaço cartesiano em configurações no espaço das juntas. O robô pode então executar a tarefa realizando movimentos "cegos", nos quais se assume que o meio envolvente se mantém estático após o início do movimento.

- **Controlo Visual**

  O termo controlo visual (*visual servoing*) foi introduzido, por Hill e Park [41] em 1979, para distinguir a nova abordagem de trabalhos anteriores. Em 1980, Sanderson e Weiss [42] introduziram a taxionomia de sistemas visuais de servos:

  1. Sistemas dinâmicos "vê-e-move" (*look-and-move*): O controlo do robô é efectuado em duas fases: o sistema de visão fornece informação ao controlador do robô, o qual utiliza a realimentação das juntas para internamente estabilizar o robô. Como é referido em [40], a maioria dos sistemas utiliza esta abordagem.
  2. Sistemas visuais de servos directos: Nestes sistemas o controlador visual calcula directamente as entradas para as juntas do robô.

A ideia geral do controlo visual é determinar a relação entre o robô e a informação sensorial para estimar o movimento necessário para minimizar o erro. O controlo visual envolve várias áreas de pesquisa, tais como, modelação de robôs (geometria, cinemática, dinâmica), sistemas de tempo-real, teoria do controlo, integração de sistemas (sensores), visão por computador (processamento de imagem, calibração da câmara). Existem várias formas de classificar os resultados [44]: baseado na configuração dos sensores, número de câmaras usado, sequência de controlo usada (2D, 3D), interpretação do meio, algoritmo de visão usado.

A utilização de realimentação visual para controlar um robô é normalmente designada *visual servoing*, controlo visual [40], termo introduzido por Hill e Park em 1979 [41]. Características visuais (baseadas em imagem) como pontos, linhas e regiões podem ser usadas para, por exemplo, permitir o alinhamento de um manipulador com um objecto. Baseada nesta informação sensorial, é gerada uma sequência de controlo. O controlo visual tem sido estudado de várias formas, desde simples tarefas de agarrar e deslocar até à actual manipulação avançada em tempo-real. Existem diferentes tipos de controlo visual: baseado em imagem, baseado em posição e controlo visual $2\frac{1}{2}$D.

Um factor fundamental para o controlo visual está relacionado com a interacção física câmara(s)-robô. O modelo de interacção câmara(s)-robô-objecto, como referido anteriormente, depende dos parâmetros da câmara e do conhecimento prévio do meio envolvente onde se encontra o objecto e ainda:

- Do número de câmaras do sistema, podendo ser definida a **visão monocular** (uma câmara) e **visão estéreo** (duas câmaras ligadas rigidamente) e ainda o **sistema de câmaras redundantes**.

- Da configuração da(s) câmara(s) relativamente ao robô, podendo a(s) câmara(s) estar(em) rigidamente ligada(s) ao elemento terminal (*end-effector*) e a olhar para o objecto, *eye-in-hand*, ou colocada(s) no espaço a olhar tanto para o elemento terminal como para o objecto, *eye-to-hand* (Fig. 4.1).

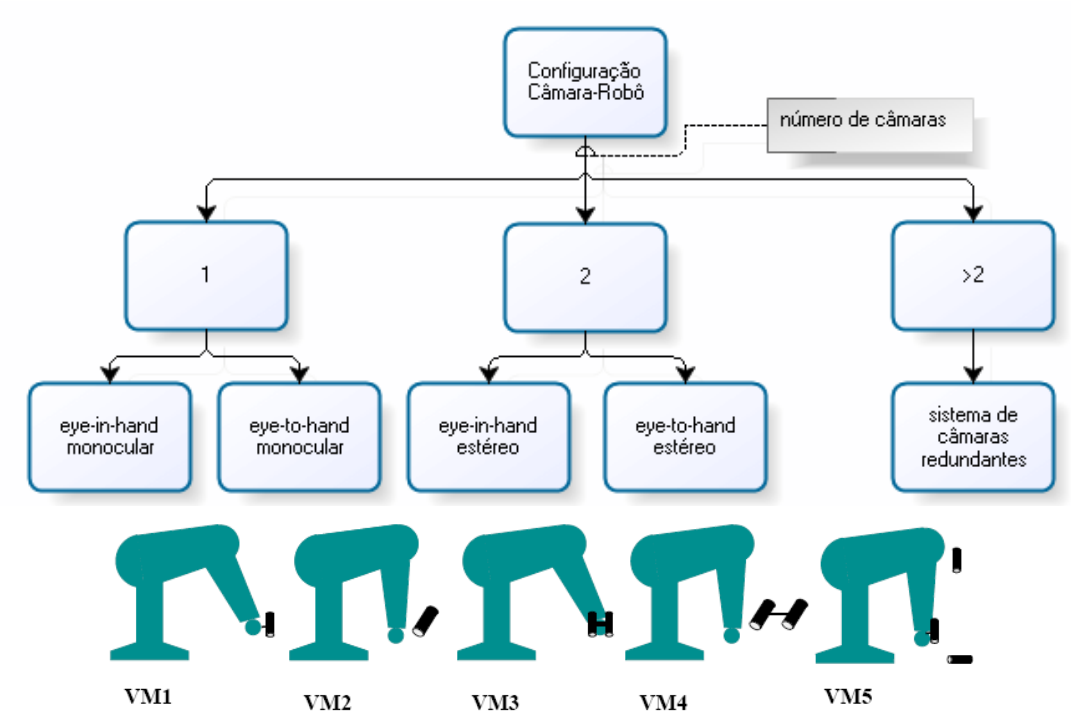


Figura 4.1: Configurações câmara-robô usadas em controlo visual (da esquerda para a direita): **VM1** *eye-in-hand* monocular, **VM2** *eye-to-hand* monocular, **VM3** *eye-in-hand* estéreo, **VM4** *eye-to-hand* estéreo, **VM5** sistemas de câmaras redundantes.

Os sistemas podem ainda ser distinguidos pelo modelo de interacção entre as coordenadas da câmara e as características do objecto na imagem (estimativa do modelo visual-motor), definindo dois grupos (Fig. 4.2). Um engloba os modelos conhecidos *a priori* através de relações analíticas e o outro inclui os modelos estimados numericamente. Conhecendo a cinemática do robô, a relação entre movimento no espaço das juntas e cartesiano pode ser determinada. Estes sistemas são classificados como sistemas em que o modelo visual-motor é conhecido *a priori*. A informação visual usada pode ser de dois tipos, i) informação visual bidimensional expressa em coordenadas do plano de imagem, ou ii) tridimensional utilizando o modelo câmara/objecto para obter informação no sistema de coordenadas câmara/robô/mundo real. Dependendo do tipo de realimentação e calibração câmara-robô, os sistemas são ainda classificados como baseados em posição, baseados em imagem ou $2\frac{1}{2}$D (híbridos).

Os sistemas baseados em imagem são normalmente preferidos sobre os baseados em posição devido à dificuldade (impossibilidade em alguns casos) de obter/manter uma boa calibração.

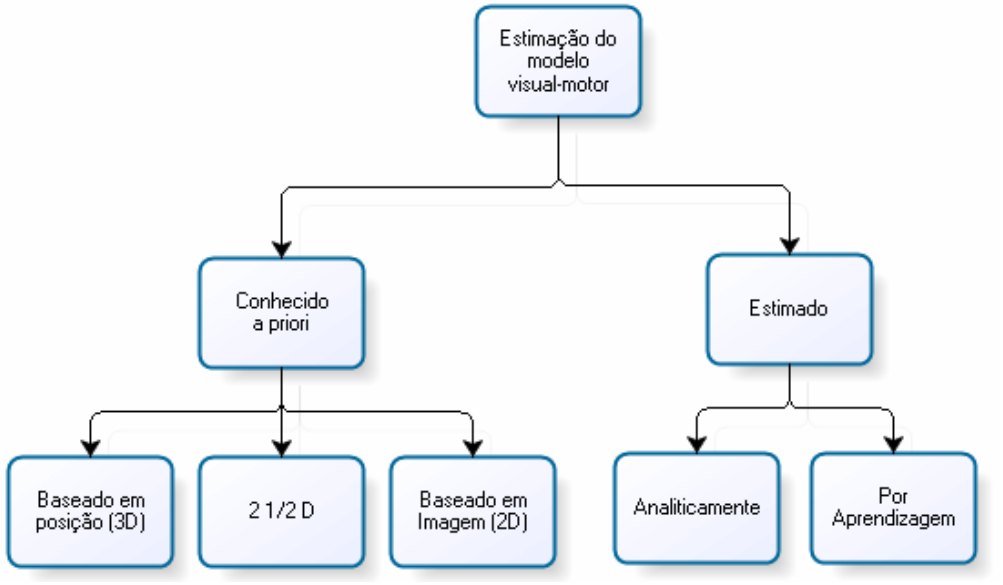


Figura 4.2: Sistemas relativamente ao modelo visual-motor. Nos sistemas onde o modelo é conhecido *a priori*, é usado o modelo cinemático do robô, parâmetros da câmara e ainda diferentes níveis de calibração entre a câmara e o robô, para estimar o movimento desejado. Existem ainda os sistemas que estimam o modelo visual-motor quer por aprendizagem ou analiticamente, permitindo o controlo sem o conhecimento da geometria do robô.

Contudo, o conhecimento da transformação entre plano de imagem e espaço das juntas continua a ser necessário. Por outro lado, existem sistemas que eliminam completamente a calibração e realizam a estimativa de um modelo visual-motor ou *on-* ou *off-line*. Este modelo pode ser estimado a) analiticamente ou b) por aprendizagem ou treino.

Controlo baseado em posição (3D)

Este tipo de controlo também é conhecido como controlo visual 3D, uma vez que a informação retirada da imagem é usada para determinar a posição e orientação do objecto relativamente à câmara ou a um ponto do meio. O erro entre a posição actual (s) e desejada (s^*) é definido no espaço cartesiano. Assim, o erro é função dos parâmetros de posição. Existem duas razões que fazem com que esta técnica, tipicamente, não seja a escolhida para tarefas de controlo visual:

1. Requer a estimativa da posição do alvo ou um modelo;

2. Para estimar a velocidade das juntas do robô e de forma a determinar um posicionamento correcto, é necessário uma calibração precisa do sistema (câmara, câmara/robô).

Um diagrama de blocos da abordagem de controlo visual baseado em posição pode ser visto na Fig. 4.3.

Em geral, a principal vantagem desta abordagem é o facto da trajectória câmara/robô ser controlada directamente em coordenadas cartesianas. Isto permite um planeamento de trajectórias mais simples para, por exemplo, evitar obstáculos. Contudo, no caso da câmara ser calibrada grosseiramente (*e.g.*, os parâmetros serem apenas aproximadamente conhecidas),

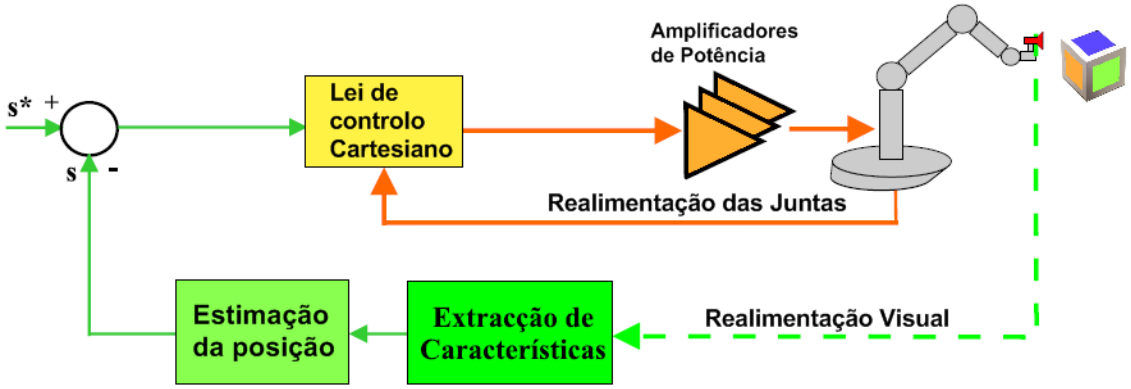


Figura 4.3: Diagrama de blocos do controlo baseado em posição

as posições actual e desejada da imagem não vão ser estimadas com precisão, o que resulta numa baixa performance (em termos de precisão) ou inclusivamente num fracasso da tarefa de controlo visual. Uma solução para este problema é projectar um sistema em malha fechada no qual tanto o alvo como o elemento terminal estão no campo visual durante a execução da tarefa [40].

Controlo baseado em imagem (2D)

O controlo visual baseado em imagem envolve o cálculo do Jacobiano de Imagem ou a matriz de interacção [40, 48, 52]. O Jacobiano de Imagem, J, representa a relação diferencial entre o plano imagem o plano da cena capturada. Os sistemas de controlo visual baseados em imagem expressam o erro directamente no espaço de imagem 2D. Se forem usadas como medida as posições na imagem das características, a função de erro é definida simplesmente como a diferença entre as posições actual ($\mathbf{f}^c$) e desejada ($\mathbf{f}^*$):

$$\mathbf{e}(\mathbf{f}) = \mathbf{f}^* - \mathbf{f}^c \tag{4.1}$$

A abordagem mais comum para gerar o sinal de controlo é a utilização de um simples controlador proporcional [57, 53].

$$\mathbf{u} = \dot{\mathbf{q}} = \mathbf{K}\mathbf{J}^{-1}\mathbf{e}(\mathbf{f}) \tag{4.2}$$

onde $\mathbf{J}^{-1}$ é a (pseudo-)inversa do Jacobiano de Imagem e $\mathbf{K}$ é uma matriz de ganho constante. A função do erro é actualizada a cada imagem adquirida e utilizada juntamente com o Jacobiano para cálculo a entrada de controlo do robô.

A estimativa do Jacobiano de Imagem requer o conhecimento dos parâmetros intrínsecos e extrínsecos da câmara. Se apenas uma câmara for usada durante o processo de controlo, perde-se informação de profundidade, esta é uma das desvantagens da abordagem. Se for usado um sistema *eye-to-hand*, e se a calibração entre o robô e a câmara é (parcialmente) conhecida, a informação de profundidade exigida para o cálculo do Jacobiano de Imagem pode ser determinada usando a cinemática directa do robô e os parâmetros de calibração. A matriz do Jacobiano de Imagem depende também do tipo de características usadas e da própria tarefa de controlo [47].

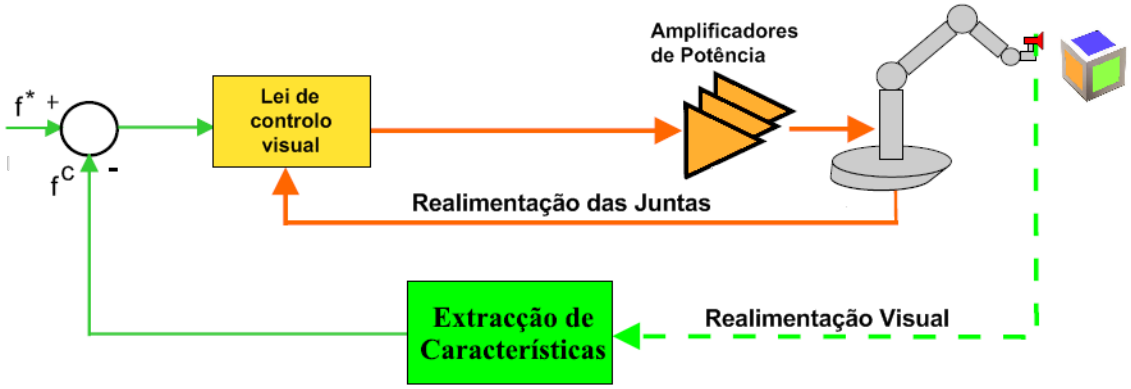


Figura 4.4: Diagrama de blocos do controlo baseado em imagem

Comparada a uma configuração *eye-in-hand* onde o objecto a ser manipulado geralmente não se encontra no campo de visão da câmara, uma configuração *eye-to-hand* permite observar facilmente tanto o objecto como o manipulador simultaneamente [47]. Se for utilizado um sistema de câmaras estéreo, a propriedade seguinte pode ser usada: zero disparidades entre um ponto no manipulador e um ponto no objecto em duas imagens significa que estes dois pontos são o mesmo ponto no espaço. Ou seja, a função de erro é minimizada simultaneamente em duas imagens. O Jacobiano de Imagem é estimado simplesmente concatenando dois Jacobianos de Imagem monoculares. O controlo visual baseado em imagem é considerado muito robusto no que diz respeito a erros da calibração da câmara e do robô [40, 58]. Uma calibração grosseira afecta apenas a taxa de convergência da lei de controlo, sendo necessário mais tempo para alcançar a posição desejada.

Controlo $2\frac{1}{2}$D (Híbrido)

A abordagem de controlo visual $2\frac{1}{2}$D foi apresentada por Malis *et al.* [56]. O método foi proposto originalmente para uma configuração *eye-in-hand*. Esta abordagem encontra-se "a meio caminho" entre as abordagens clássicas baseadas em posição e baseadas em imagem. Evita as suas respectivas desvantagens: contrariamente à baseada em posição, não necessita de um modelo geométrico 3D do objecto. Comparando com o controlo visual baseado em imagem, assegura a convergência da lei de controlo no espaço da tarefa.

4.3 Aquisição e Processamento de Imagem

O sistema de visão é constituído por uma câmara a cores, uma unidade *pan-tilt*, a Unidade de Processamento (PC/104) e as unidades Master/Slave associadas ao controlador de baixo nível das juntas (Fig. 4.5). Após a aquisição de imagem, a Unidade de Processamento faz o pré-processamento da imagem com uma redução de resolução (*pyr down*). Esta imagem é então segmentada, por cor, no espaço HSV, e a máscara resultante é utilizada para calcular o centro de massa do objecto no plano de imagem. Por fim, a informação é usada como entrada para um algoritmo de controlo baseado em imagem, do qual resulta uma sequência de controlo que é comunicada aos servomotores de forma a realizar o seguimento do nosso objecto alvo,

a bola. Esta secção é dedicada aos aspectos relacionados com a aquisição e processamento de imagem, sendo as questões de geração da sequência de controlo tratadas na próxima secção.

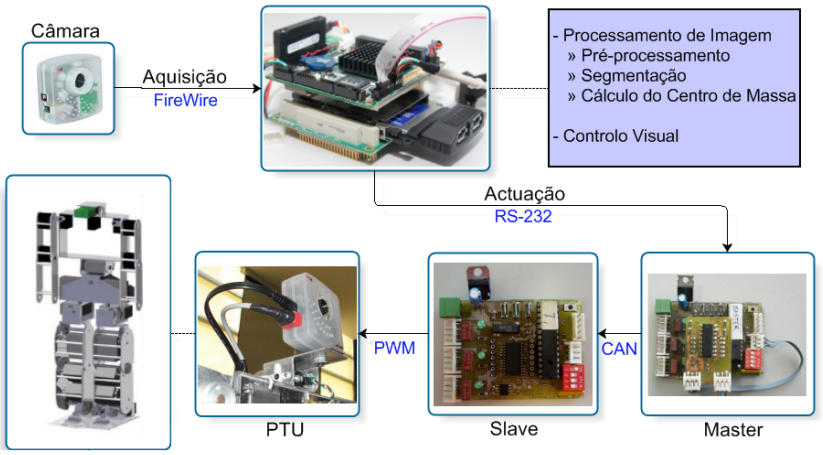


Figura 4.5: Diagrama representativo do Sistema de Visão

4.3.1 Aquisição de Imagem

A captura do sinal de vídeo é feita através de uma câmara UniBrain Fire-i @ 30fps (640x480) (Fig. 4.6) ligada a uma placa PCMCIA UniBrain FireCard (Fig. 4.7) que faz a ponte PCMCIA-FireWire. Todo o processamento de imagem é posteriormente efectuado em tempo-real na Unidade de Processamento.

Figura 4.6: UniBrain Fire-i

Figura 4.7: UniBrain FireCard

Na Figura 4.8 pode ser visto o módulo PC/104-Plus com a placa PCMCIA-FireWire.

A aquisição por software começou por ser feita com recurso às funções pertencentes à biblioteca OpenCV (versão I), mas esta foi abandonada pois era apenas possível fazer aquisição à resolução 640x480, o que introduzia atrasos bastantes grandes no processo de aquisição e no processamento da imagem. Mesmo optando por realizar uma redução de resolução (*pyr down*) para 160x120, que é uma resolução suficiente para efectuar a localização por cor, era necessário efectuar esta operação duas vezes, 640x480 -> 320x240 -> 160x120. Embora se ganhasse tempo no processamento, ao trabalhar com imagens com um quarto da resolução, gastava-se muito tempo com operações *pyr down* e também recursos de processamento. Pode ser verificado que o tempo médio de execução de um ciclo de controlo ascendia a 125ms (∼70ms em aquisição e *pyrdown*), o que significa que temos uma *framerate* de 8Hz, o que é manifestamente pouco

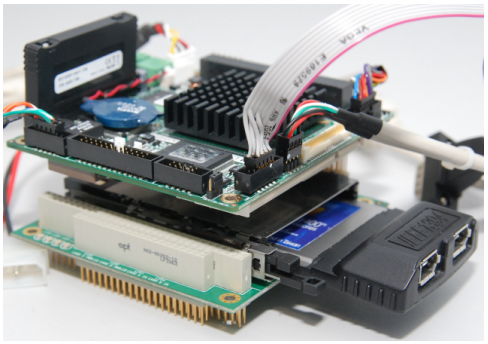


Figura 4.8: Módulo de expansão PCMCIA e placa adaptadora FireWire

para acompanhar alvos com uma certa dinâmica, como é o caso da bola no terreno de jogo (Tabela 4.1).

Tempos de Execução (ms)				
	máximo	**mínimo**	**médio**	**desvio**
aquisição	31,0230	11,8310	20,2630	4,6492
pyr down	83,6740	47,1840	47,6000	3,2164
segmentação	21,9830	9,5830	9,9274	1,0795
centro de massa	3,1960	3,1050	3,1327	0,0136
controlo	0,0170	0,0140	0,0148	0,0062
comunicação	52,8640	3,0700	43,1530	12,5260
total	156,0990	80,1240	124,0900	13,3760

Tabela 4.1: Tempos de Execução - versão I (aquisição - biblioteca do OpenCV) - 128 observações

Foi decidido utilizar para a aquisição uma biblioteca alternativa que efectua a aquisição directa por DMA recorrendo às directivas do sistema e não ao OpenCV. Desta forma tornou-se possível escolher a resolução desejada na aquisição e assim poupar tempo. Pode-se verificar na Tabela 4.2 que o tempo de aquisição é bastante reduzido e consequentemente o tempo total médio de execução, que é reduzido para aproximadamente um quarto do anterior, ∼28ms, conseguindo-se explorar o framerate máximo da câmara (30fps).

Contudo, a aquisição não era correctamente efectuada na resolução escolhida, pelo que se passou a adquirir a imagem à resolução de 320x240 (Fig. 4.9) e de seguida efectuar apenas um *pyr down* para a resolução de 160x120. Assim, conseguiu-se obter uma imagem correctamente adquirida e a um tempo inferior ao inicial, 35ms (tendo em conta a redução de resolução) (Tabela 4.3). Verifica-se então que o tempo total de execução passa a ∼52ms, correspondentes a um framerate de 19Hz, um valor mais razoável para o acompanhamento de alvos móveis.

4.3.2 Segmentação de Cor

O principal desafio de todo o processo de visão prende-se com a correcta identificação do objecto em causa seja qual for o meio e as condições de funcionamento; ora esta é uma tarefa

Tempos de Execução (ms)				
	máximo	**mínimo**	**médio**	**desvio**
aquisição	6,4660	2,8860	3,1088	1,9602
pyr down	-	-	-	-
segmentação	42,1950	9,7330	10,3318	2,7850
centro de massa	18,8870	2,7850	2,9364	1,0359
controlo	0,2530	0,0140	0,0168	0,0161
comunicação	3637,2710	2,4100	11,7372	12,5260
total	3653,0450	18,6400	28,1310	162,6566

Tabela 4.2: Tempos de Execução - versão II (aquisição - biblioteca proprietária) - 499 observações

Figura 4.9: Imagem Adquirida da Câmara

Tempos de Execução (ms)				
	máximo	**mínimo**	**médio**	**desvio**
aquisição	33,6860	12,0350	13,8940	2,1626
pyr down	1783,1240	9,4760	18,6820	125,7200
segmentação	169,9590	9,7050	10,9220	11,4220
centro de massa	11,4430	3,0680	3,4089	0,8132
controlo	0,2350	0,0140	0,0164	0,0157
comunicação	90,4820	2,0150	4,5410	6,3831
total	1816,1850	39,4350	51,4640	126,6200

Tabela 4.3: Tempos de Execução - versão III (biblioteca proprietária c/ *pyr down*) - 199 observações

complexa pois as condições que nos permitem identificar um objecto são inúmeras, dimensões, forma, cor e até outros mais complexos como a sua textura e posição no meio envolvente podem ser utilizados para fazer esta distinção. O meio utilizado para identificação do objecto neste trabalho foi baseado na distinção por cor. Esta escolha foi tomada uma vez que na competição

do *RoboCup* tudo se baseia em código de cores: balizas de cores diferentes (azul e amarelo)[1], campo verde, linhas brancas e bola cor-de-laranja de forma a criar um maior contraste com o campo. No entanto, a identificação por cor pode tornar-se uma tarefa exigente dado que luz e movimento podem causar dificuldades [24].

A segmentação de cor é feita pegando na imagem já com menor resolução(Fig. 4.10) e convertendo-a para o espaço de cores HSV.

Figura 4.10: Imagem após redução de resolução

De seguida a imagem convertida é separada nas componentes H (*Hue* - Tonalidade), S (*Saturation* - Saturação) e V (*Value* - Valor)(Fig. 4.11) e a estas componentes são aplicados limiares inferiores e superiores de modo a filtrar componentes de cor indesejados.

Figura 4.11: Componentes H, S e V com limiares aplicados

É então efectuada a recombinação das componentes formando assim a máscara de cor(Fig. 4.12) que vai ser aplicada à imagem(Fig. 4.13).

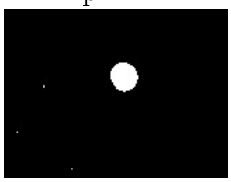


Figura 4.12: Máscara de cor

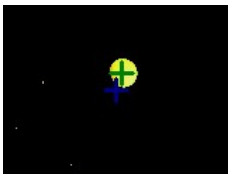


Figura 4.13: Imagem final

Os limiares das componentes H, S e V referidos anteriormente podem ser ajustados mediante a execução da aplicação no modo de calibração, recorrendo às respectivas janelas, como se pode ver na Fig. 4.14.

4.3.3 Localização do Objecto no Plano de Imagem

Após a segmentação de cor é necessário proceder à localização do objecto no plano de imagem. A localização é feita calculando o centro de masssa do objecto no plano de imagem para que se possa usar essa informação posteriormente para fazer o seu seguimento. Usando a

[1] Actualmente, as balizas já não distinguíveis pela cor, são ambas brancas com rede tal como as reais. Isto deve-se à tentativa de aproximar a competição dos moldes do futebol humano.

Figura 4.14: Janelas de calibração das componentes H, S e V

máscara, que é uma imagem binária onde 0 é preto e 1 é branco, pode-se calcular o centro de massa através do número de *pixels* brancos em cada ordenada da imagem. O algoritmo que faz este cálculo é relativamente simples e consiste na criação de duas imagens temporárias mx e my, com os seus *pixels* inicializados a zero, que servirão para armazenar as coordenas xx e yy, respectivamente, dos *pixels* brancos (1) da máscara. De seguida percorrem-se todas as colunas de todas as linhas da máscara e caso o *pixel* seja branco armazena-se no *pixel* correspondente em mx e my, os valores das coordenadas xx e yy, respectivamente. No final calculam-se os valores médios das imagens mx e my que correspondem às coordenadas xx e yy do centro de massa da máscara (objecto).

$$mx(i,j) = i, \quad \forall mask(i,j) = 1 \tag{4.3}$$

$$my(i,j) = j, \quad \forall mask(i,j) = 1 \tag{4.4}$$

$$CM_x = \frac{\sum_{i=0}^{120}\sum_{j=0}^{160} mx(i,j)}{160 \times 120} \tag{4.5}$$

$$CM_y = \frac{\sum_{i=0}^{120}\sum_{j=0}^{160} my(i,j)}{160 \times 120} \tag{4.6}$$

Como é de esperar, a localização do objecto pode ser afectada por interferências externas pois mesmo que a calibração esteja muito boa, poderá sempre existir nas redondezas um outro objecto que partilhe a cor com o nosso alvo. Assim, assumindo que é feito uma localização inicial válida, passou-se a definir uma região de interesse (ROI) em torno do objecto-alvo (Fig. 4.15). Esta ROI é uma área quadrada com centro coincidente com o centro de massa calculado para o objecto e dimensão variável, dependendo a dimensão da relação entre a área da ROI e da área do objecto (Fig. 4.16). A utilização deste método além garantir alguma imunidade a interferências externas reduz significativamente o tempo médio de processamento (Tabela 4.4), pois após uma ROI estar definida apenas essa porção da imagem é usada no processamento. A dimensão variável da ROI, quadrado de lado $2\delta_i$, garante um refinamento constante da zona de busca. Assim, consegue-se um desempenho ainda melhor do ciclo total de controlo, sendo que o *framerate* ascende a $\sim$25Hz. Pode no entanto acontecer, que a bola saia dos limites da ROI; quando se dá este caso, a ROI é redefinido como a totalidade da imagem e a bola é novamente procurada. Ao encontrar novamente a bola, uma nova ROI é definido e a localização volta a ser feita baseada na ROI. No caso extremo da bola sair completamente da imagem, a PTU executa um varrimento ao longo dos seus limites físicos, de modo a localizá-la.

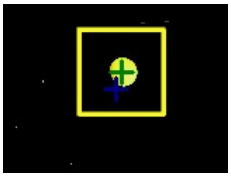


Figura 4.15: Imagem com ROI em torno do objecto

Tempos de Execução (ms)				
	máximo	**mínimo**	**médio**	**desvio**
aquisição	32,4020	11,8780	13,6820	2,0275
pyr down	25,9050	9,4730	9,8432	1,6330
segmentação	41,6030	9,3320	9,8456	2,4185
centro de massa	3,2550	0,3970	1,3079	0,4478
controlo	0,1590	0,0140	0,0154	0,0093
comunicação	37,2460	2,1670	4,4850	2,6913
total	118,6600	35,9060	39,1520	7,1468

Tabela 4.4: Tempos de Execução - versão IV (c/ ROI dinâmica) - 244 observações

4.4 Posicionamento e Seguimento Visual

Um robô humanóide enfrenta desafios complexos quando num ambiente como um jogo de futebol e dependendo quase exclusivamente da informação visual para controlar as suas acções. É imperativo localizar a bola que poderá estar em qualquer posição do terreno de jogo, ou mesmo a saltar. Não esquecendo que o alvo em movimento tem uma trajectória que é impossível de prever. Torna-se então, necessário efectuar a detecção visual e o respectivo alinhamento com a bola, sempre que esta sai do campo de visão. Assim sendo, a PTU recebe ordens para executar um varrimento por forma a procurar a bola nos limites físicos das juntas.

Nesta linha de ideias, as funcionalidades desejadas para o sistema podem ser resumidas como se segue: o sistema de visão deve ser capaz de identificar a bola na cena, alinhar a câmara com o objecto alvo controlando de forma apropriada a unidade de *pan-tilt* e manter a atenção sobre o objecto seleccionado. Desta forma, as tarefas de controlo com realimentação visual em análise neste trabalho são: (i) posicionamento: alinhar a câmara com o alvo; e (ii) seguimento: manter uma relação constante entre a câmara e o alvo em movimento.

Esta secção descreve a implementação da arquitectura de realimentação visual, centrando-se na estrutura de controlo e na questão do sinal de erro. Por um lado, é usada uma estrutura de controlo hierárquica com o sistema de visão a fornecer as referências de entrada para o controlador ao nível das juntas do robô (controlo de baixo-nível). O plano de imagem é também usado para obter uma estimativa da posição relativa da bola em relação ao sistema de coordenadas base do robô. A restricção física de todos os objectos relevantes estarem localizados no plano do chão permite ter uma correspondência de um-para-um entre um ponto do plano do chão e seu correspondente no plano de imagem.

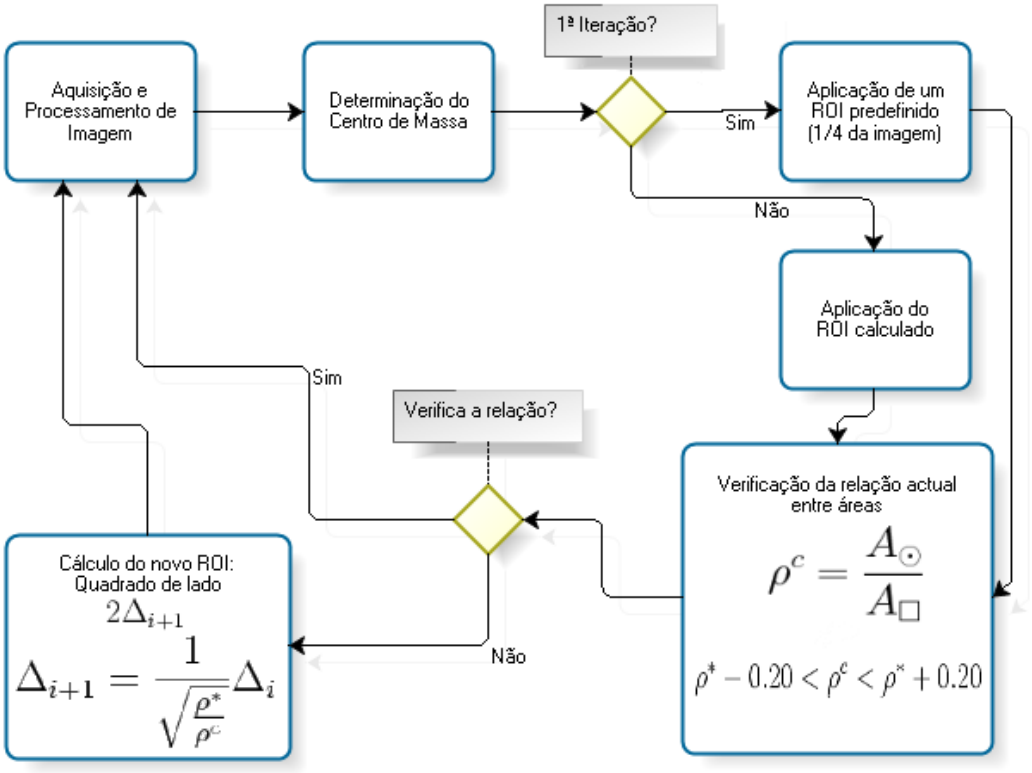


Figura 4.16: Algoritmo de definição da região de interesse

4.4.1 Estimativa da Localização da Bola

Nesta subsecção descreve-se a forma como a informação do plano de imagem é usada para estimar a posição da bola no sistema de coordenadas 3D do robô. Assume-se que todos os objectos relevantes estão localizados no solo, i.e., as coordenadas segundo z_R são nulas. A restrição imposta permite obter uma correspondência de um-para-um entre um ponto no plano do solo e a sua observação no plano de imagem. Para realizar as tarefas desejadas torna-se importante conhecer e entender a cinemática envolvida. Para o efeito, considere-se a estrutura cinemática ilustrada na Fig. 4.17, onde se representam os sistemas de coordenadas de referência S_R (com origem na base do robô) e da câmara S_C. Usando o algoritmo de Denavit-Hartenberg [37], obteve-se a matriz de transformação homogénea que descreve a posição e orientação da câmara em relação ao sistema de coordenadas de referência:

$$ {}^{\mathbf{R}}\mathbf{T_C} = \left[\begin{array}{ccc|c} -S_1 & C_1S_2 & C_1C_2 & l_2C_1C_2 + l_3C_1S_2 \\ C_1 & S_1S_2 & S_1C_2 & l_2S_1C_2 + l_3S_1S_2 \\ 0 & C_2 & -S_2 & -l_2S_2 + l_3C_2 + l_0 + lt_H + l_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.7} $$

em que C_i e S_i são as funções coseno e seno do ângulo i, respectivamente, q_1 é o ângulo de *pan*, q_2 o ângulo de *tilt*, e os l's são os comprimentos dos elos (ver Fig. 4.17).

Em primeiro lugar, considera-se a situação particular em que a câmara está alinhada com

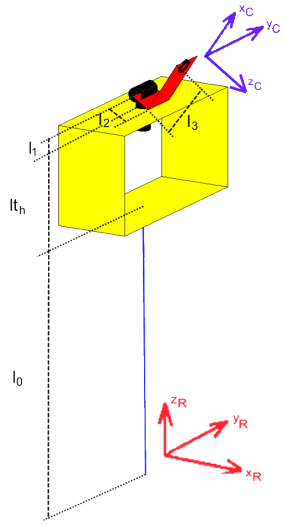


Figura 4.17: Estrutura cinemática do sistema em análise

o alvo. Neste caso, define-se um raio colinear com o versor $\hat{z}_C$ que parte do centro óptimo da câmara e cuja intersecção no solo permite obter uma estimativa da posição da bola em S_R. As equações de cinemática directa que relacionam a posição da bola com os ângulos das juntas são dadas por:

$$\begin{cases} x_B = \frac{C_1}{S_2}(l_3 + LC_2) \\ y_B = \frac{S_1}{S_2}(l_3 + LC_2) \end{cases} , L = l_0 + lt_h + l_1 \tag{4.8}$$

No caso mais geral, assumindo a mesma restrição, as novas coordenadas são obtidas a partir dos processos geométricos que determinam a imagem:

$$\begin{cases} x_B = \frac{[L-(l_2+\lambda)S_2+l_3C_2]S_1x_C+(\lambda+l_2-LS_2)C_1y_C+(l_3+LC_2)\lambda C_1}{\lambda S_2+C_2y_C} \\ y_B = \frac{-[l_3C_2-(l_2+\lambda)S_2+L]C_1x_C+[l_2+\lambda-LS_2]S_1y_C+(l_3+LC_2)}{\lambda S_2+C_2y_C} \end{cases} \tag{4.9}$$

em que x_C e y_C são as coordenadas da imagem (na conversão entre as coordenadas em pixel e as coordenadas da imagem ignora-se a distorção radial da lente) e λ é a distância focal da lente. Estas equações foram implementadas e testadas, podendo vir a ser usadas em desenvolvimentos futuros sempre que seja necessário obter uma estimativa da localização da bola.

4.4.2 Algoritmo de Controlo Baseado na Imagem

A relação espacial entre o alvo e a câmara é estimada directamente no plano de imagem e o vector de erro expresso em termos das características da imagem. Mais concretamente, a imagem adquirida é processada de forma a obter a posição do centro de massa em relação ao centro do plano de imagem, i.e., o desvio da bola. O objectivo do controlo é manter o centro de massa da bola o mais próximo possível do centro da imagem de vídeo, ao mesmo tempo que se reduz o atraso computacional e erros devidos à calibração da câmara e/ou modelação dos sensores. A escolha da relação entre os ângulos das juntas e o desvio da bola permite

determinar algoritmos com diferentes desempenhos. O método mais simples é a aplicação directa de uma lei proporcional tal como:

$$\delta q = Ke \tag{4.10}$$

Aqui, $\delta q = [\delta q_1, \delta q_2]^T$ é o vector de incremento das juntas, K é uma matriz diagonal de ganhos constantes e $e = [x_C, y_C]^T$ é o vector de erro definido pelo desvio da bola. Esta abordagem possui a vantagem da sua simplicidade: cada componente do vector de erro relaciona-se directamente e de uma forma independente com as juntas *pan* e *tilt*. Contudo, nesta abordagem, veio a verificar-se ser difícil usar os mesmos ganhos do controlador quando a bola se deslocava devagar/depressa e quando se encontrava longe/perto do robô.

De forma a ultrapassar esta dificuldade, é proposta uma lei de controlo alternativa para realizar as tarefas de posicionamento e seguimento baseada na adopção de ganhos variáveis. A ideia básica é identificar, de forma experimental, o comportamento do sistema em diferentes cenários de forma a estabelecer uma relação entre os ganhos do controlador e a configuração actual da PTU. Para determinar de forma empírica esta relação, foram realizados movimentos simples da bola e analisada a evolução dos erros no espaço da câmara (usando ganhos fixos). Uma constatação imediata foi a dificuldade do ângulo de *pan* (q_1) em efectuar o seguimento sempre que a bola realiza uma trajectória perpendicular ao plano sagital do robô, tanto mais acentuado quanto mais próxima se encontra a bola do robô.

Assumindo que a orientação da câmara se encontra restringida, de tal forma que aponta para o solo em todos os casos, o projecto do controlador de ganhos variáveis reduz-se à seguinte expressão:

$$\delta q = \left[\frac{K_p x_C}{\cos(q_2)} \quad , \quad \frac{K_t y_C}{\sin(q_2)} \right]^T \tag{4.11}$$

Mais uma vez, para uma dada posição da bola, as coordenadas no plano de imagem, x_C e y_C, apenas afectam os ângulos *pan* e *tilt*, respectivamente. Contudo, os termos do denominador podem ser entendidos como ganhos adaptativos, compensando o atraso no seguimento. De referir que qualquer dos métodos propostos não requer a formulação de uma calibração explícita entre o plano de imagem e o sistema de coordenadas do mundo.

Uma vez calculados os valores a atribuir às juntas da plataforma, a sequência de controlo é enviada para a unidade Master que por sua vez a transmitirá à unidade Slave correspondente. A unidade Slave é então responsável por implementar o controlo local (controlador de baixo nível das juntas) gerando um sinal de PWM que actua nos servomotores (Fig. 4.18). A malha de realimentação do controlador local funciona a 50 Hz (20ms).

Este sistema de controlo visual não utiliza realimentação directa das juntas do robô pelo facto da informação obtida dos potenciómetros não se ter revelado muito fiável. Em substituição foi utilizada uma "realimentação virtual" por *software* baseada nas posições de referência enviadas para os servos.

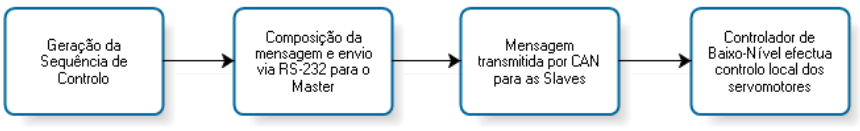


Figura 4.18: Processo de actuação na plataforma humanóide

4.5 Apresentação e Discussão de Resultados

Para avaliar o desempenho do sistema de visão, efectuaram-se várias experiências com a Unidade de Processamento a funcionar a 500MHz com 512MB de *RAM*. A experiência envolve uma câmara com um CCD a cores e uma cabeça *pan-tilt* situada no topo de um robô humanóide que se mantém num posição estática de pé durante a totalidade da execução de cada ensaio.

A Tabela 4.5 mostra os limites dos tempos de execução do processamento visual, cálculos de controlo para o seguimento visual e comunicação com a unidade Master. Os servomotores *pan* e *tilt* são controlados por uma unidade Slave que recebe os comandos enviados pela Unidade de Processamento para a unidade Master que controla o barramento CAN.

Algoritmo	Máx.(ms)	Min.(ms)	Média(ms)	Desv. Pad.(ms)
Aquisição	32.4	11.9	13.7	2.0
Pyr Down	25.9	9.5	9.8	1.6
Segmentação/Filtragem	41.6	9.3	9.9	2.4
Localização Centro de Massa	3.3	0.6	1. 3	0.5
Comunicação	37.4	2.2	4.5	2.7

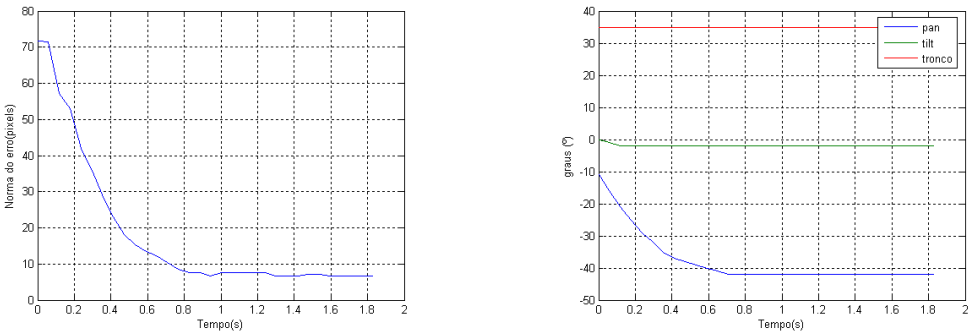
Tabela 4.5: Limites superiores nos tempos de execução dos processos do processamento de visão na Unidade de Processamento (244 observações).

Relativamente aos algoritmos de processamento descritos acima, é expectável encontrar uma variação considerável na execução temporal do processo, uma vez que, em alguns casos, a bola é encontrada de imediato ou, em outros casos, é necessário analisar toda a imagem (*e.g.*, quando a bola desaparece do campo visual). Contudo, o tempo médio de execução do processo ronda os 40ms. Isto significa que as imagens a cores adquiridas pela câmara são processadas a uma cadência aproximada de 25Hz, o que é suficiente para permitir estímulos de movimentos rápidos ou outro tipo de informação visual de variação rápida.

De forma a verificar a implementação e avaliar o desempenho dos algoritmos propostos, foram realizadas várias experiências. Os erros de seguimento no plano de imagem calculados como a norma do desvio da bola, estão ilustrados para o alinhamento e para o seguimento, nas Fig. 4.19 e 4.20 respectivamente. Como pode ser observado dos eixos do erro de seguimento, o controlador atinge o alinhamento em aproximadamente 1s, enquanto que os erros de seguimento chegam a 20 *pixels* quando a bola está em movimento. O movimento corresponde a uma deslocação perpendicular ao plano sagital do robô, com $z = 0$. Nesta experiência foi utilizado o controlador com ganhos variáveis.

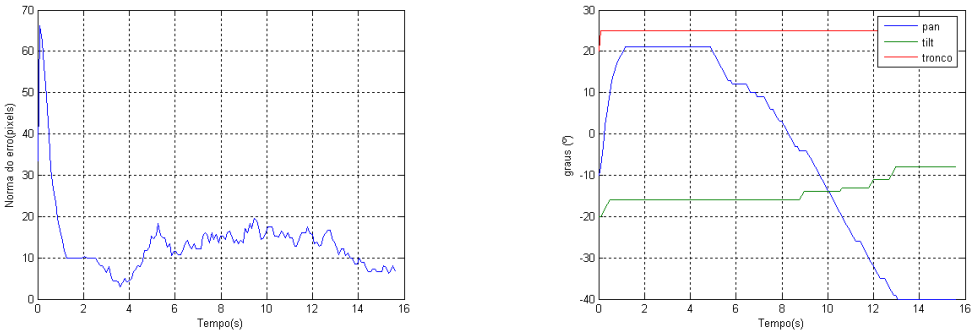
Na Fig. 4.21 ilustra-se o comportamento do robô realizando o mesmo movimento com o controlador de ganhos fixos e a evolução do sinal de controlo das juntas. É perceptível que o aumento do erro na zona correspondente à passagem da bola pela frente do robô. Nessa zona o controlador proporcional com ganhos fixos não é capaz de acompanhar o deslocamento da bola, enquanto que o controlador com ganhos variáveis compensa ligeiramente o erro do seguimento nessa zona. No entanto, a diferença não é suficientemente acentuada que permita tirar conclusões definitivas sobre os métodos.

Foi também realizada uma experiência de seguimento usando apenas o deslocamento vertical do robô, ou seja, os graus de liberdade *tilt* da PTU e do tronco. Os resultados encontram-se na Fig. 4.22. Verifica-se que o erro associado é semelhante ao obtidos nas situações anteriores. O erro toma valores mais elevados a partir da entrada em funcionamento do grau de liberdade



(a) (b)

Figura 4.19: Resultados experimentais do alinhamento visual em termos da norma do desvio da bola usando o controlador com ganhos variáveis: (a)a câmara alinha-se com a bola; e (b) evolução do sinal de controlo das juntas.

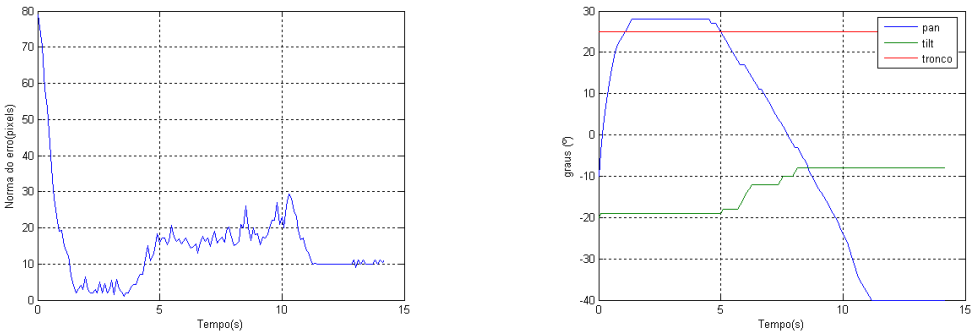


(a) (b)

Figura 4.20: Resultados experimentais do seguimento visual em termos da norma do desvio da bola usando o controlador com ganhos variáveis: (a)a bola desloca-se ao longo de uma linha recta com uma velocidade média de 0.2 m/s.; e (b)evolução do sinal de controlo das juntas.

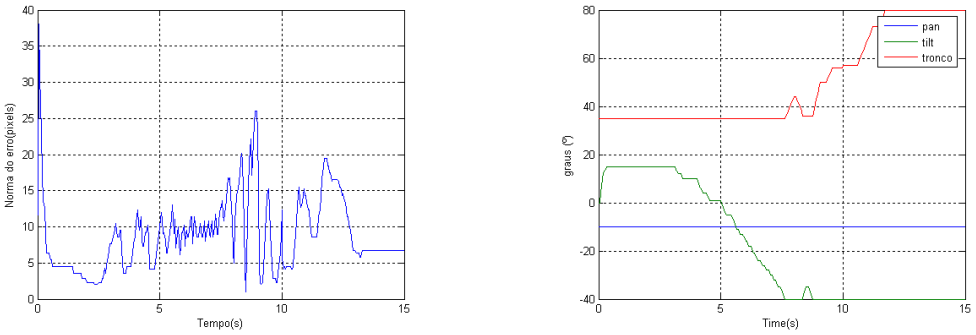
do tronco. Estes resultados apontam para a necessidade de aprofundar a questão da utilização do grau de liberdade do tronco e o seu respectivo controlo.

Uma diminuição dos erros de seguimento pode ser alcançada aumentando os ganhos do controlador, o que torna os resultados de robustez, em todo a área de trabalho, pobres de um ponto de vista prático. De modo a aumentar a aplicabilidade prática da abordagem, é necessária aprofundar a pesquisa corrente para evitar o paradigma de altos ganhos como único meio de reduzir os erros de seguimento.



(a) (b)

Figura 4.21: Resultados experimentais do seguimento visual em termos da norma do desvio da bola o controlador com ganhos fixos: (a)a bola desloca-se ao longo de uma linha recta com uma velocidade média de 0.2 m/s; e (b)evolução do sinal de controlo das juntas.



(a) (b)

Figura 4.22: Resultados experimentais do seguimento visual em termos da norma do desvio da bola: (a)erro associado; e (b) evolução do sinal de controlo das juntas.

4.6 Sistema de Visão em Tempo-Real

4.6.1 Enquadramento

No âmbito do desenvolvimento do sistema de visão do Humanóide foi considerada e estudada a hipótese de este ser feito com restrições temporais. Pretendia-se estudar o impacto desta abordagem relativamente a outra sem tais preocupações e avaliar os respectivos desempenhos.

Um sistema de tempo-real é um sistema que desempenha uma tarefa que tem de ser prestada num intervalo de tempo finito imposto por um qualquer processo físico. O resultado da computação de um sistema tempo-real deve produzir resultados a tempo e lógicamente correctos. O meio envolvente com o qual o sistema interage possui um ritmo de evolução próprio, o qual é inerente ao processo físico e não pode ser alterado externamente, isto é tempo-real. É importante notar que tempo-real não significa rapidez, mas sim um ritmo de evolução próprio.

Para a implementação do sistema é necessária a utilização das funcionalidades de tempo-real uma vez que as tarefas associadas a este sistema têm de ser executadas dentro de um intervalo de tempo finito, não interessa processar imagens capturadas há muito tempo pois o meio envolvente já se alterou entretanto. Para o efeito foi utilizada a biblioteca *PMan*[25] desenvolvida no âmbito do Projecto CAMBADA[2]. Para a comunicação entre processos foram utilizadas memórias partilhadas (*shared memories*), reguladas por semáforos.

Para efeitos de estudo foi utilizada uma versão monolítica, com código sequencial em ciclo infinito, e uma versão multi-tarefa, em que cada tarefa é um processo *per si* dependendo apenas da informação partilhada. O estudo das duas possibilidades, devido às exigentes necessidades de *debug*, foi realizado num computador Pentium M @ 1,73GHz, 1Gb RAM, com Ubuntu 7.10 (kernel 2.6.22-14-generic).

Podemos então resumir os objectivos deste estudo nos seguintes pontos:

1. Definição e modularização do processo.
2. Escalonamento Tempo-Real.
3. Discussão de outros processos do Humanóide.
4. Comparação com uma implementação Monolítica.

♦ **PMan**

Os *GPOSs*[3] não possuem capacidades que são tipicamente necessárias por sistemas embebidos, por exemplo, suporte a activação automática de tarefas recorrentes com precisão suficiente, controlo de fase e restrições de precedência. Estas dificuldades foram percepcionadas no âmbito de equipa de futebol robótico CAMBADA, desenvolvida na Universidade de Aveiro e levaram ao desenvolvimento de uma biblioteca de gestão de processos no espaço de utilizador, chamada *PMan*, que confere qualidade de serviço ao GPOS em utilização, Linux no caso presente.

4.6.2 Definição e Modularização do Processo

A arquitectura definida para o sistema é a que está representada na Fig. 4.23. Esta consiste em quatro tarefas, Image Acquisition, Image Processing, Object Tracking e Control. A primeira tarefa faz o *trigger* do sistema, ou seja, após a aquisição da imagem, as tarefas seguintes são despertadas e executam a sua função.

A tarefa Image Aquisition é, como já foi referido, o *tick* do sistema, activa os processos seguintes quando uma nova imagem está disponível. A imagem adquirida é armazenada numa shared memory (SHM_ATOP) para posterior utilização por parte de outros processos (Fig. 4.24).

Image Processing recolhe da shared memory referida anteriormente (SHM_ATOP) a imagem armazenada e efectua o respectivo processamento de cor, a imagem em RGB é convertida para HSV e posteriormente dividida nas suas componentes H, S e V. Os parâmetros, previamente ajustados com a tarefa auxiliar Color Calibration, são então aplicados a cada

[2]CAMBADA (Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture) - Equipa de Futebol Robótico Liga Média da Universidade de Aveiro, http://www.ieeta.pt/atri/cambada/index.htm

[3]*GPOS - General Purpose Operating System*, Sistemas Operativos de Uso Geral

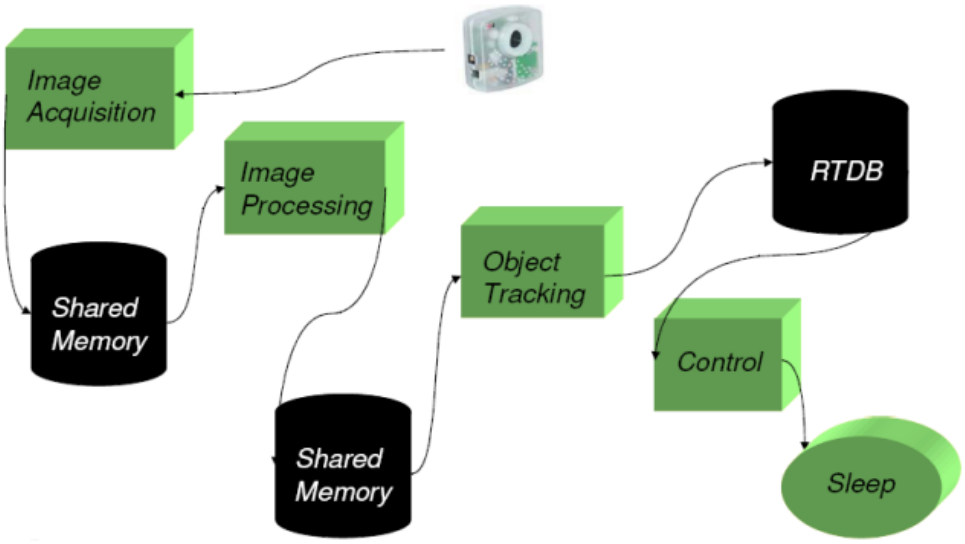


Figura 4.23: Arquitectura do Sistema de Visão

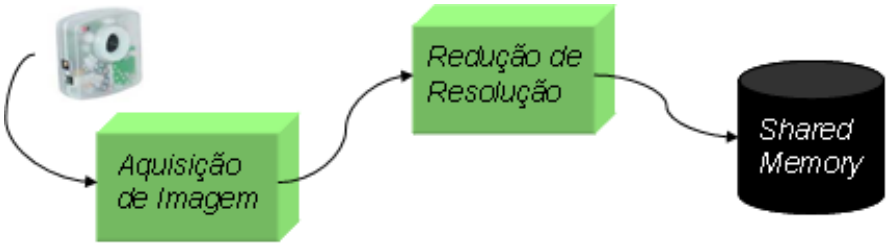


Figura 4.24: Tarefa de Aquisição de Imagem

componente e estas são de seguida combinadas formando a máscara. Esta máscara é então armazenada numa outra shared memory (SHM_PTOT) (Fig. 4.25).

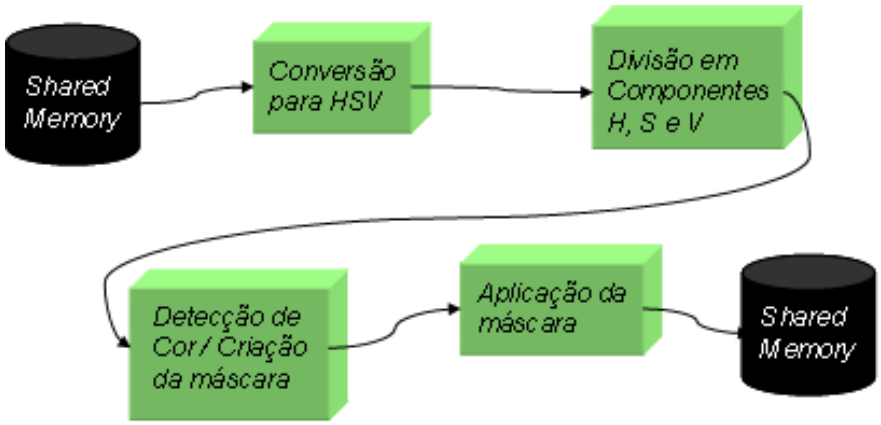


Figura 4.25: Tarefa de processamento de imagem

Object Tracking faz a detecção da bola na imagem e respectiva validação através da de-

tecção de círculos na imagem filtrada. É calculado o centro de massa do objecto detectado e determinada a sua localização relativa na imagem, estes parâmetros são então armazenados numa base de dados tempo-real (SHM_RTDB), também uma memória partilhada. A sequência dos eventos do processamento de imagem está representada na Fig. 4.26. A Fig. 4.27

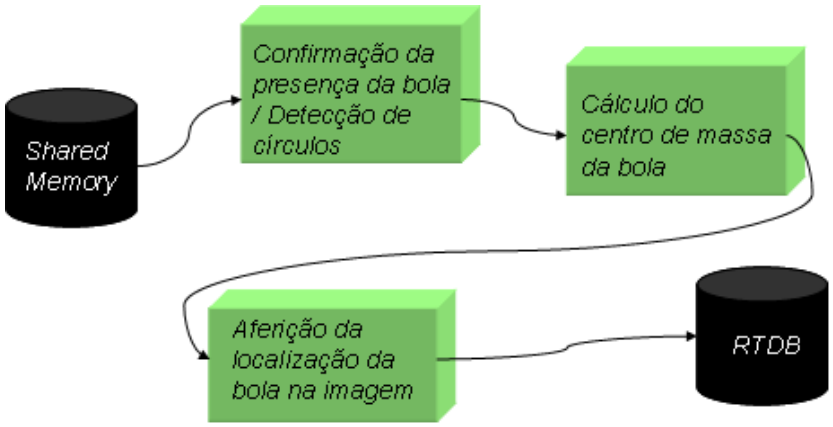


Figura 4.26: Tarefa de localização da bola

ilustra os passos desde a aquisição da imagem até à determinação dos parâmetros do objecto, passando criação da máscara através das componentes H, S e V da imagem.

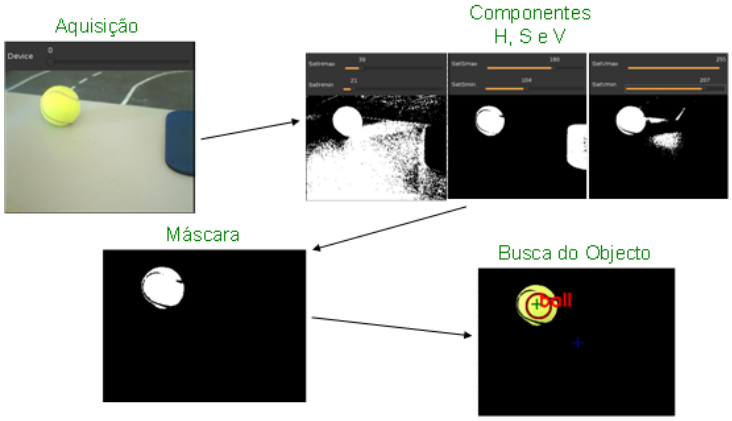


Figura 4.27: Sequência de processamento de imagem

A tarefa Control recolhe da RTDB, os parâmetros calculados pela tarefa Object Tracking e faz uma validação dos valores. De seguida, são calculados os parâmetros do movimento a efectuar, por exemplo, uma deslocação da bola no plano da imagem, corresponde a uma combinação de movimentos das juntas do pescoço necessários para que o humanóide possa seguir a bola com a cabeça (câmara). Para determinar os parâmetros do movimento foi utilizado um algoritmo proporcional (Fig. 4.28).

Color Calibration é um processo autónomo interactivo e separado do sistema de controlo, que permite obter os parâmetros que serão posteriormente utilizados no processamento de imagem. A calibração é feita com recurso a janelas com barras de deslocamento que fazem

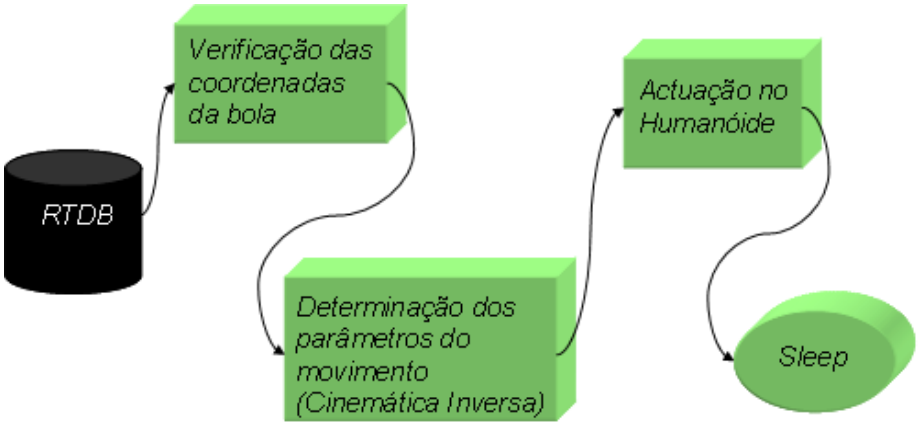


Figura 4.28: Tarefa de actuação e controlo do Humanóide

o ajuste "on-line" de cada componente (Fig. 4.29). No final da execução os parâmetros são salvos para um ficheiro de texto chamado 'hsv_param.txt'.

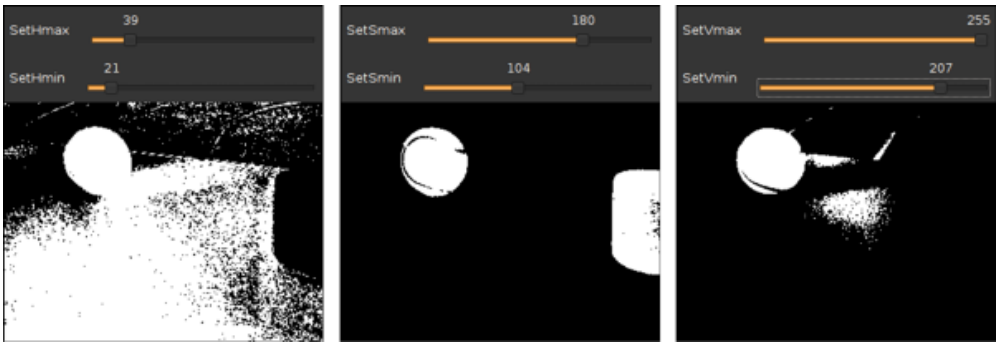


Figura 4.29: Janelas de calibração

Foi depois necessário definir precedências para as tarefas, como está ilustrado na Fig. 4.30. A sequência das tarefas é lógica: captura da imagem, processamento da mesma, extracção dos parâmetros do objecto alvo e finalmente actuação no humanóide concordante com a informação recolhida e tarefa a executar que, no caso específico, é seguir uma bola.

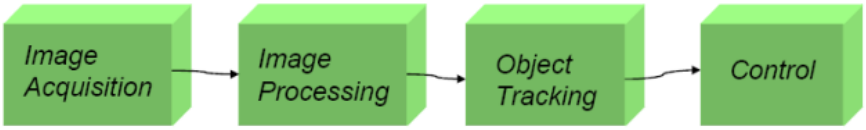


Figura 4.30: Esquema de precedências

Uma vez que não existem tarefas concorrentes poderá não fazer muito sentido falar em prioridades. Apenas existindo uma tarefa a executar essa é sempre prioritária. No entanto pode ser definido um *esqueleto* de prioridades de tarefas com base na importância da tarefa no processo global, e também tendo em vista uma implementação futura.

As prioridades foram atribuídas usando o critério "tarefa predecessora, maior prioridade".

É de notar que em Linux maior prioridade significa menor valor, ou seja, quanto mais próximo de 0 (zero) for o valor da prioridade de uma tarefa, mais prioritária ela se torna.

A Tabela 4.6 contém um resumo da caracterização das tarefas relativamente ao escalonamento.

Processo	**Período**	**Lista de Precedências**	**Prioridade**	**Descrição**
Image Acquisition	1	-	-	Interface com a câmara.Tick do sistema.
Image Processing	1	*Image Acquisition*	30	Classificação de cor.
Object Tracking	1	*Image Processing*	40	Busca do objecto (bola).
Control	1	*Object Tracking*	50	Execução do controlo no Humanóide.

Tabela 4.6: Classificação e descrição das tarefas

4.6.3 Apresentação e Discussão de Resultados

Verifica-se na versão monolítica que o tempo médio de execução de cada ciclo (≈67ms) (Tabela 4.7) é cerca do dobro do tempo que demora a adquirir uma nova imagem (30 fps@(640x480) ≈ 33ms entre aquisições) (Fig. 4.31).

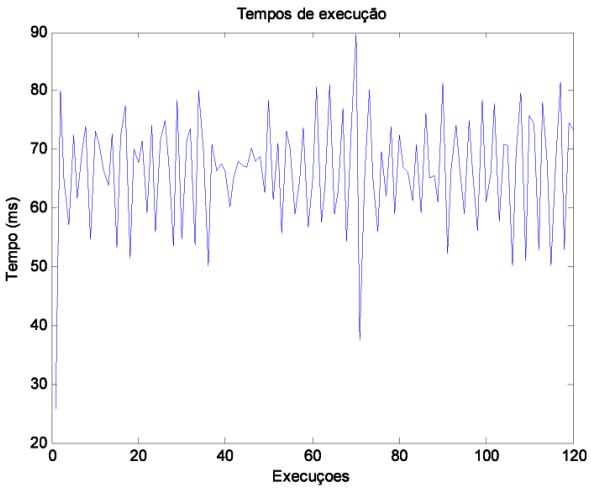


Figura 4.31: Execução da Versão Monolítica

Passando para a versão multi-tarefa verificamos que o tempo médio de execução da tarefa de aquisição é semelhante ao tempo da versão monolítica e os tempos das tarefas subsequentes são

Tempos de Execução (ms)	
Mínimo	25,822
Máximo	89,599
Médio	66,4272
Desvio Padrão	9,6604

Tabela 4.7: Tempos relativos à execução da versão monolítica (128 observações)

muito inferiores quando comparados com o anterior. A diferença que se obtém entre as versões monolítica e multi-tarefa aponta para uma pior performance da versão multi-tarefa. Isto pode ser explicado devido a mudanças de contexto e preempções originadas pela concorrência entre tarefas, e ainda acessos a zonas de recursos partilhadas(Fig. 4.32 e Tabela 4.8).

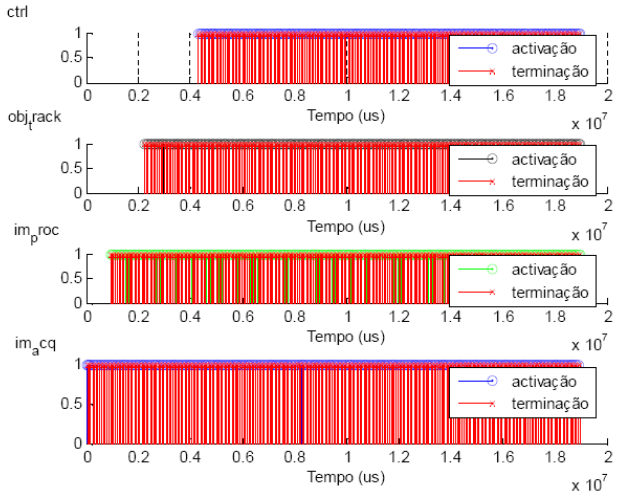


Figura 4.32: Evolução temporal das tarefas

Tempos de Execução (ms)				
	im_acq	im_proc	obj_track	ctrl
Mínimo	55,8850	6,1850	0,0000	0,0045
Máximo	78,0790	9,5440	0,5510	0,0057
Médio	66,5484	6,3977	0,1649	0,0030
Desvio Padrão	3,8843	0,3613	0,2148	0,083

Tabela 4.8: Tempos de execução relativos à execução individual de cada uma das tarefas do processo (250 observações)

Podemos inferir que no estado actual de implementação do Humanóide, que executa um único processo: procura e seguimento da bola, a solução multi-tarefa não será vantajosa pois não é tirado partido das propriedades de gestão de processos concorrentes, definições de prioridades e peridiocidades entre eles. É necessário estudar futuramente o impacto desta imple-

mentação quando se partir para a execução de múltiplos processos concorrentes, procura da bola, baliza, linhas, etc.

Capítulo 5

Conclusões

5.1 Discussão de Resultados

O grande objectivo de qualquer plataforma humanóide é atingir a total autonomia energética e computacional. Este trabalho visava a integração da Unidade Central de Processamento, de forma a fornecer-lhe autonomia de processamento. A inclusão deste controlador PC/104 baseado em Linux revelou-se um bom compromisso entre capacidade de processamento, versatilidade, consumo e tamanho. É agora possível efectuar todo o processamento na plataforma, com a integração da Unidade de Processamento, na arquitectura distribuída de controlo já existente. A Unidade de Processamento serviu de base à criação de um ambiente de desenvolvimento versátil que pode ser utilizado para desenvolvimento directo na plataforma ou em alternativa, este ser feito externamente e posteriormente descarregado para a plataforma. O acesso remoto à plataforma via rede *Ethernet* é também uma alternativa.

Muitas aplicações robóticas dependem do uso de um sistema de visão baseada numa câmara, capaz de garantir um sentido de percepção do meio envolvente. No âmbito deste trabalho, o ênfase foi dado ao projecto de um esquema de controlo baseado em visão que permitisse a operação autónoma de um robô humanóide. A arquitectura implementada separa o processamento de visão alto-nível, do controlo baixo-nível, em malha fechada dos actuadores. Como consequência, foi possível criar dois métodos para o controlo do seguimento global. Cada um dos controladores "fecha o ciclo" no plano de imagem e está provado ser globalmente válido. O sistema de visão atingiu um nível apreciável de funcionamento com a taxa média de processamento do ciclo de controlo a ser aproximadamente 25Hz, um número suficiente para permitir estímulos rápidos e outros tipos de entradas visuais com variação rápida. Estes resultados tornam emergente a necessidade de avançar para a utilização de algoritmos de controlo mais robustos. A utilização de uma região de interesse dinâmica permite obter uma maior imunidade ao ruído causado por elementos "estranhos" ao controlo. O processo de localização e alinhamento quando a bola se encontra numa posição fixa é bastante satisfatório exibindo valores de erro relativamente baixos, e rápida convergência a partir do momento em que a bola se encontra no campo visual. Estes resultados são relevantes para esta classe de robôs humanóides de baixo custo e baseados em componentes comerciais.

5.2 Trabalho Futuro

A realização deste trabalho permitiu identificar vários pontos que necessitam de ser melhorados e também algumas perspectivas de trabalho futuro:

- No que concerne ao processamento de imagem torna-se necessário avançar para a detecção de forma para validar o que é detectado por cor. Isto permitirá uma melhor exactidão da busca da bola.

- Outro passo é o reconhecimento de outros elementos, por exemplo, os presentes no terreno de jogo da competição *RoboCup*. Poderá ser interessante, nessa altura, estudar novamente a solução com restrições temporais, abordada neste documento, pois a introdução de tarefas paralelas requererá uma maior atenção aos tempos de processamento e partilha de recursos.

- Relativamente ao controlo visual torna-se interessante explorar a utilização do Jacobiano da imagem, e ainda testar o efeito de um controlador PI (ou outros mais robustos) no desempenho do sistema.

- Outro assunto em aberto é a influência do movimento do robô na informação visual e no desempenho do sistema de seguimento. Por exemplo, uma questão a aprofundar é se, e até que ponto, o sistema de visão distingue movimento do alvo e o seu próprio movimento.

Apêndice A

Tutorial de Instalação USB

Este tutorial detalha a instalação do Linux por este método. É de referir que o mesmo pode ser aplicado a qualquer computador que possua na BIOS, a opção de arranque por dispositivos USB. Para a instalação do Linux através de um dispositivo USB deve-se criar a imagem de *boot*. Assim, é necessário um PC com Linux instalado e acesso à Internet[1] para se poder criar a imagem a imagem de boot.

Assume-se neste tutorial que o dispositivo USB está a funcionar em /dev/sda, a localização exacta pode ser verificada com o comando **cat /proc/partitions** numa *shell*. Assume-se ainda que o dispositivo USB se encontra vazio pois o procedimento leva a perda de dados contidos no mesmo.

1. Primeiro procede-se ao *unmount* do dispositivo USB:

   sudo umount /dev/sda

2. De seguida faz-se o download da imagem boot.img.gz necessária para arrancar com o dispositivo USB (*bootable*):
   wget ftp://ftp.debian.org/debian/dists/stable/main/installer-i386/current/images/hd-media/boot.img.gz

3. Extrai-se a imagem e grava-se no dispositivo USB:

   sudo zcat boot.img.gz > /dev/sda

   Nota: *Neste caso está a ser usada uma ISO do* Debian Net-install. *Pode ser usada também uma versão completa ou ainda um Iso* Business Card *ou ainda uma outra distribuição. No entanto deve garantir que usa a image de boot adequada.*

4. Monta-se o volume em /mnt:

   sudo mount /dev/sda /mnt

[1] Alternativamente, os ficheiros necessários podem ser descarregados noutro PC e transferidos para o PC com Linux.

5. E efectua-se o download da ISO, correspondente à distribuição que pretendemos instalar, para o dispositivo USB:

   cd /mnt/
   sudo wget http://cdimage.debian.org/debian-cd/4.0_r0/i386/iso-cd/
   debian-40r0-i386-netinst.iso

6. Pode-se agora fazer o *unmount* do volume:

   sudo umount /dev/sda

7. Remove-se então o dispositivo USB do sistema.

8. Configura-se a BIOS do PC alvo para efectuar o arranque a partir de dispositivos USB e insere-se o mesmo. De seguida reinicia-se o PC e procede-se à instalação conforme as instruções apresentadas no ecrã.

   Nota: *Toda a instalação do Linux efectuada no Humanóide foi feita segundo os parâmetros standard exceptuando o Particionamento do Disco. Aqui não foi criada área de SWAP, sendo a totalidade do disco reservada para o sistema de ficheiros (ext3). A razão para esta opção será abordada na secção 3.7.*

Apêndice B

Configuração da Rede

Os comandos apresentados de seguida são executados em modo normal se precedido por $ ou em modo de superutilizador quando precedidos por #.

Na versão instalada no Humanóide, o comando **sudo**[1] não estava disponível (**sudo** não instalado), assim para aceder ao modo de superutilizador é necessário fazer o login na *shell*, **$su**.

1. Primeiramente é necessário abilitar o *inetd*[2] no arranque:

   [root]# vim /etc/rc.conf
   adicionar a linha ao ficheiro:
   inetd = "YES"

2. De seguida é necessário configurar a interface de rede ethernet, neste caso **eth0**:

   [root]# vim /etc/network/interfaces
   e adicionar (editar) as linhas correspondentes à interface
   auto eth0
   iface eth0 inetd dhcp

3. Para acesso à rede exterior à Universidade de Aveiro é ainda necessário configurar a *proxy* de rede:

   - No ficheiro **/etc/environment**

     [root]# vim /etc/environment
     adicionar as linhas:
     http_proxy = "http://proxy.ua.pt:3128"
     ftp_proxy = "http://proxy.ua.pt:3128"
     no_proxy = "localhost,.ua.pt"

[1] sudo (super user do), é um programa para sistemas operativos Unix que permite aos utilizadores executarem comandos e correrem programas com privilégios de segurança de outros utilizadores (normalmente o superutilizador)

[2] inetd é um super-servidor *deamon*(processo especial que corre em segundo plano) que existe nos sistemas Unix e gere os serviços de rede.

Acquire{
HTTP::Proxy proxy.ua.pt:3128;
FTP::Proxy proxy.ua.pt:3128;
} ;
alias wget = “wget -Y on”

[root]# vim /etc/profile
adicionar a linha:
export http_proxy ftp_proxy no_proxy

Apêndice C

Pacotes adicionados - Instalação e Configuração

C.1 Ferramentas de Sistema

C.1.1 Sudo

Instalação

[root]# apt-get install sudo

Configuração

É agora necessário configurar quais os utilizadores que podem obter privilégios de outros utilizadores, neste caso o utilizador *phua-user* obter privilégios de *root* (super utilizador). Para isso é necessário editar o ficheiro /**etc**/**sudoers**:

[root]# visudo -f /etc/sudoers

adicionando a linha correspondente ao utilizador pretendido, este passa a poder obter os privilégios de *root*:

phua-user ALL=(ALL) ALL

A partir de agora passa a ser possível executar qualquer comando que só pudesse ser invocado pelo *root* precedendo a o mesmo pela palavra chave **sudo** e inserindo a *password* quando solicitada.

C.1.2 Xfce

Instalação

Além do Xfce é necessário instalar a base do gestor de janelas, o servidor X.

[root]# apt-get install x-window-system-core
[root]# apt-get install xfce4

Configuração

O Xfce não necessita de configuração prévia.

C.1.3 Coriander

Instalação

[root]# apt-get install coriander

Configuração

Coriander não necessita de qualquer configuração.

C.2 Ferramentas de Edição e Compilação de Código-Fonte

C.2.1 Editor - Vim

Instalação

[root]# apt-get install vim

Configuração

O *Vim* não necessita de configuração para iniciar a sua utilização, no entanto existem algumas que podem ajudar a melhorar o desempenho da edição. Para proceder à configuração deve-se editar/criar o ficheiro ∼/**.vimrc**[1]:

[root]# vim ∼/.vimrc

Adicionar cada uma das linhas abaixo tem o efeito descrito.

- **:set incsearch**
  *a busca passa a ser incremental, ou seja, o texto introduzido passa a ser procurado como palavra **ou** parte de palavras.*

- **:set ignorecase**
  a busca deixa de ser sensível a maiúsculas/minúsculas

- **:set smartcase**
  a opção smartcase serve de complemento a ignorecase, se ambas tiverem activas, o Vim vai ignorar maiúsculas/minúsculas apenas se o padrão de busca estiver todo em minúsculas. Se existir alguma maiúscula no padrão então o Vim fará a busca de acordo.

[1] ∼ quando usado em caminhos significa "o directório *home* do utilizador actual". Assim, o utilizador *root* executar **cd** ∼ é equivalente a executar **cd /home/root/**

- **:set wildmode=longest,list**
  Esta opçao do wildmode faz com que o Vim reaja da seguinte forma. Ao introduzir parte de um nome de ficheiro e premir TAB, o Vim completa o nome com a alternativa mais longa de entre todas as existentes. Se seguida aguarda por uma das acções seguintes: ENTER para confirmar o nome do ficheiro, prosseguir na escrita do nome do ficheiro, ESC para cancelar o comando, ou TAB novamente. Premir TAB uma segunda vez faz com que o Vim exiba uma lista com todos os ficheiros possíveis que poderiam completar o nome parcial inserido.

- **:set scrolloff=n**
  *O **n** significa que se deseja manter pelo menos **n** linhas de contexto em redor do cursor em todas as alturas. O Vim deslocará automaticamente o ficheiro de modo a que o cursor não fique próximo do topo ou do fundo do ecrã mais que **n** linhas.*

- **:syntax on**
  activa o sintaxe de cores, o Vim detecta qual o tipo de ficheiro que está a editar e usa um esquema de cores apropriado.

C.2.2 Compilador - GCC

Instalação

[root]# apt-get install gcc

Configuração

O GCC não necessita de qualquer configuração prévia.

C.2.3 Gestão de Compilação - Make

Instalação

[root]# apt-get install make

Configuração

O Make não necessita de configuração prévia.

C.2.4 Ambiente Integrado de Desenvolvimento - KDevelop

Instalação

[root]# apt-get install kdevelop

Foram ainda instaladas as aplicações **automake** e **autoconf** para o seu correcto funcionamento.

[root]# apt-get install automake
[root]# apt-get install autoconf

Configuração

O KDevelop não necessita de configuração prévia.

C.3 Bibliotecas de Programação

C.3.1 OpenCV

Instalação

[root]# apt-get install libcv-dev
[root]# apt-get install libcvaux-dev
[root]# apt-get install opencv-doc

Configuração

A biblioteca OpenCV não necessita de qualquer configuração.

Apêndice D

Tutorial de Criação de uma *RAM disk*

1. Antes de começar deve-se verificar o que existe por defeito no sistema. O Linux cria por defeito 16 *RAM disks*, no entanto estas não estão "activas" (não usam qualquer *RAM*). São listados os dispositivos ram0 - ram19, mas apenas ram0 - ram15 são, por defeito, passíveis de utilização. Para verificar estes dispositivos usa-se o seguinte comando:

   [root]# ls -l /dev/ram*
   lrwxrwxrwx 1 root root 4 Jun 12 00:31 /dev/ram -> ram1
   brw-rw—- 1 root disk 1, 0 Jan 30 2003 /dev/ram0
   brw-rw—- 1 root disk 1, 1 Jan 30 2003 /dev/ram1
   brw-rw—- 1 root disk 1, 10 Jan 30 2003 /dev/ram10
   brw-rw—- 1 root disk 1, 11 Jan 30 2003 /dev/ram11
   brw-rw—- 1 root disk 1, 12 Jan 30 2003 /dev/ram12
   brw-rw—- 1 root disk 1, 13 Jan 30 2003 /dev/ram13
   brw-rw—- 1 root disk 1, 14 Jan 30 2003 /dev/ram14
   brw-rw—- 1 root disk 1, 15 Jan 30 2003 /dev/ram15
   brw-rw—- 1 root disk 1, 16 Jan 30 2003 /dev/ram16
   brw-rw—- 1 root disk 1, 17 Jan 30 2003 /dev/ram17
   brw-rw—- 1 root disk 1, 18 Jan 30 2003 /dev/ram18
   brw-rw—- 1 root disk 1, 19 Jan 30 2003 /dev/ram19
   brw-rw—- 1 root disk 1, 2 Jan 30 2003 /dev/ram2
   brw-rw—- 1 root disk 1, 3 Jan 30 2003 /dev/ram3
   brw-rw—- 1 root disk 1, 4 Jan 30 2003 /dev/ram4
   brw-rw—- 1 root disk 1, 5 Jan 30 2003 /dev/ram5
   brw-rw—- 1 root disk 1, 6 Jan 30 2003 /dev/ram6
   brw-rw—- 1 root disk 1, 7 Jan 30 2003 /dev/ram7
   brw-rw—- 1 root disk 1, 8 Jan 30 2003 /dev/ram8
   brw-rw—- 1 root disk 1, 9 Jan 30 2003 /dev/ram9
   lrwxrwxrwx 1 root root 4 Jun 12 00:31 /dev/ramdisk -> ram0

2. De seguida, **grep** através da saída de **dmesg** para descobrir qual o tamanho das *RAM disks* actuais:

   [root]# dmesg | grep RAMDISK
   RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize

Como se pode ver o tamanho por defeito de uma RAM disk é 4 MB. Pretende-se no Humanóide implementar uma RAM disk com metade da RAM total, 512 MB, então o próximo passo é configurar o Linux para usar uma RAM diskmaior no arranque.

3. Aumentar o tamanho da *RAM disk*:
   O tamanho das *RAM disks* é controlado por opção de linha de comandos que é passada ao *kernel* durante o arranque. Uma vez que o GRUB é o *bootloader*[1] para o Debian, deve-se alterar o ficheiro de configuração **/boot/grub/menu.lst** com as nova opçao do *kernel*. A opção para o *kernel* para o tamanho das *RAM disks* é:
   ramdisk_size=xxxxx, onde xxxxx é o tamanho expresso em blocos de 1024 bytes. Isto é o que se deve adicionar ao ficheiro para configurar uma *RAM disk*de 128MB:

   ```
   # ...
   ### ## End Default Options ##

   title Debian GNU/Linux, kernel 2.6.18-4-486
   root (hd0,0)
   kernel /boot/vmlinuz–2.6.18–4–486 root=/dev/hda1 ro ramdisk_ size=128000
   initrd /boot/initrd.img–2.6.18–4–486
   savedefault

   title Debian GNU/Linux, kernel 2.6.18–4–486 (single-user mode)
   root (hd0,0)
   kernel /boot/vmlinuz-2.6.18–4–486 root=/dev/hda1 ro single
   initrd /boot/initrd.img–2.6.18–4–486
   savedefault

   ### END DEBIAN AUTOMAGIC KERNELS LIST
   ```

4. Depois de gravar as alterações no ficheiro, é necessário fazer o *reboot*(reiniciar) ao sistema. A seguir ao *reboot* executando novamente o comando **dmesg** podem-se confirmar as alterações efectuadas:

   ```
   [root]# dmesg | grep RAMDISK
   RAMDISK driver initialized: 16 RAM disks of 128000K size 1024 blocksize
   ```

5. Formatar a *RAM disk*
   Não existe necessidade de formatar a *RAM disk* como sistema de ficheiro com *journaling*[2], assim usa-se simplesmente o sistema de ficheiros **ext2**. Vai apenas ser usada uma *RAM disk*, assim será formatado apenas **/dev/ram0**:

   ```
   [root]# mke2fs -m 0 /dev/ram0
   mke2fs 1.40 (09-Nov-2006)
   Filesystem label=
   ```

[1] Gestor de Arranque
[2] Consultar tópico anterior, Journaling

OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
32000 inodes, 128000 blocks
0 blocks (0.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
16 block groups
8192 blocks per group, 8192 fragments per group
2000 inodes per group
Superblock backups stored on blocks:
8193, 24577, 40961, 57345, 73729
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 34 mounts or 180 days,
whichever comes first. Use tune2fs –c or –i to override.

*A opção **-m 0** previne que o comando **mke2fs** reserve espaço no sistema de ficheiros para o utilisador root, que é o seu comportamento por defeito. Pretende-se que a totalidade da RAM disk esteja disponível para o desenvolvimento do trabalho.*

6. Criar um ponto de montagem
   Com a *RAM disk* formatada é agora necessário criar um ponto de montagem para ela. De seguida é possível montar a *RAM disk* e usá-la. Para este efeito criou-se a directoria **/mnt/rd**.

   [root]# mkdir /mnt/rd [root]# mount /dev/ram0 /mnt/rd

7. Agora pode verificar-se o ponto de montagem:

   [root]# mount | grep ram0
   /dev/ram0 on /mnt/rd type ext2 (rw)
   [root]# df –h | grep ram0
   /dev/ram0 122M 1.6M 120M 2% /mnt/rd

   Pode ainda fazer-se uma análise mais detalhada com o comando **tune2fs**:

   [root]# tune2fs –l /dev/ram0
   tune2fs 1.40-WIP (14-Nov-2006)
   Filesystem volume name: <none>
   Last mounted on: <not available>
   Filesystem UUID: d349e358–ab5a–4963-8499–f18818f1da4d
   Filesystem magic number: 0xEF53
   Filesystem revision #: 1 (dynamic)
   Filesystem features: resize_inode dir_index filetype sparse_super
   Filesystem flags: signed directory hash
   Default mount options: (none)

Filesystem state: not clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 32000
Block count: 128000
Reserved block count: 0
Free blocks: 122405
Free inodes: 31989
First block: 1
Block size: 1024
Fragment size: 1024
Reserved GDT blocks: 256
Blocks per group: 8192
Fragments per group: 8192
Inodes per group: 2000
Inode blocks per group: 250
Filesystem created: Thu Nov 29 14:28:42 2007
Last mount time: Thu Nov 29 14:46:25 2007
Last write time: Thu Nov 29 14:46:25 2007
Mount count: 1
Maximum mount count: 34
Last checked: Thu Nov 29 14:28:42 2007
Check interval: 15552000 (6 months)
Next check after: Tue May 27 15:28:42 2008
Reserved blocks uid: 0 (user root)
Reserved blocks gid: 0 (group root)
First inode: 11
Inode size: 128
Default directory hash: tea
Directory Hash Seed: 5bce1045–6b82–45a9–b781–3cfaf175cf14

8. É agora necessário dar ao utilisador permissões de escrita na *RAM disk*, neste caso, o utilisador *phua-user*. Para isso é preciso mudar o dono e as permissões da directoria **/mnt/rd**:

   [root]# chown phua-user:root /mnt/rd
   [root]# chmod 0770 /mnt/rd
   [root]# ls -ald /mnt/rd
   drwxrwx— 3 phua-user root 1024 Nov 29 11:09 /mnt/rd

9. Usar a *RAM disk*

   Agora é possível copiar, mover, apagar, editar e listar ficheiros na *RAM disk* exactamente como numa partição comum.

10. Para desmontar a *RAM disk*, é simplesmente necessário o comando:

    [root]# umount -v /mnt/rd /dev/ram0 umounted

Nota: Se se voltar a montar a RAM disk, os dados ainda lá estarão. Após a alocação de memória para a RAM disk, esta é marcada de forma a que kernel não a tente usar mais tarde. Desta forma, a RAM não pode ser libertada se não necessitarmos mais da RAM disk. É por isso ser cuidadoso e não alocar mais memória para a RAM disk do que aquela estritamente necessária. Evidentemente é sempre possível libertar a memória com um reboot.

11. Criação automática de uma *RAM disk*

    Quando é necessário criar e montar uma RAM disk cada vez que o sistema inicia, é possível automatizar o processo descrito anteriormente adicionando alguns comandos ao script de inicialização ***/etc/rc.local****:*

    # Formata, monta, e define permissões numa *RAM disk* de 128MB
    /sbin/mke2fs -q -m 0 /dev/ram0
    /bin/mount /dev/ram0 /mnt/rd
    /bin/chown phua-user:root /mnt/rd
    /bin/chmod 0770 /mnt/rd

Referências

[1] Sakagami, Y. et al. (2002): *The Intelligent ASIMO: System Overview and Integration*, in Proceedings of the IEEE International Conference on Intelligent Robots and Systems, pp. 2478-2483.

[2] Kaneko, K. et al. (2004): *Humanoid Robot HRP-2*, in Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1083-1090.

[3] Hu, L. & Zhou, C. (2007) : *Locomotion planning of humanoid robot Robo-Erectus Senior (RESr-1)*, in Proceedings of the IEEE-RAS International Conference on Humanoid Robots, Nov.29 - Dez.01, Pittsburgh, USA.

[4] Furuta, T. et al. (2001): *Design and Construction of a Series of Compact Humanoid Robots and Development of Biped Walk Control Strategies*, Robotics and Automation Systems, 37, pp. 81-100.

[5] Kim, J.-H. et al. (2004): *Humanoid Robot HanSaRam: Recent Progress and Developments*, Journal of Computational Intelligence, 8(1): 45-55.

[6] Silva, F. M. & Santos, V. M. (2007): *Multipurpose Small-Cost Humanoid Platform and Modular Control Software Development*, capítulo da publicação Humanoid Robots: Human-like Machines, [ISBN: 978-3-902613-07-3], Matthias Hackel, pp. 65-88.

[7] Gameiro, D.; Carvalho, F. (2004): *Concepção e Desenvolvimento de Componentes Modulares para Controlo de uma Plataforma Robótica Antropomórfica.*, Relatório Interno, DEMUA-GAR.

[8] Rego, L.; Barbosa, R. (2004): *Estudos funcionais de uma plataforma para um sistema robótico humanóide*, Relatório Interno, DEMUA-GAR.

[9] Santos, Vítor M. F. & Silva, Filipe M. T. (2005): *Development of a Low-Cost Humanoid Robot: Components and Technological Solutions*, proceedings of the CLAWAR 2005.

[10] Gomes, L. & Silva, M. (2005): *Concepção e Desenvolvimento de Unidades de Percepção e Controlo para um Robot Humanóide*, Relatório Interno, DEMUA-GAR.

[11] Beça, N. & Ângelo Cardoso, Â. (2005): *Desenvolvimento e Integração das Sub-estruturas Inferior e Superior para a Locomoção de uma Plataforma Humanóide*, Relatório Interno, DEMUA-GAR.

[12] Santos, V. & Silva, F.M. (2005): *Engineering Solutions to Build an Inexpensive Humanoid Robot Based on a Distributed Control Architecture*, Proc. of 5th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS2005, pp.86-91, Tsukuba, Japan, 5-7 Dec.

[13] Ruas, M.; Ferreira, P.; Silva, F. M. & Santos, V. M. (2007): *Local-level Control of a Humanoid Robot Prototype with Force-driven Balance*, in Proceedings of the IEEE/RSJ International Conference on Humanoid Robots, Nov.29 - Dec.3, Pittsburgh, USA.

[14] Unibrain (2006): *Fire-i 1394 Board Camera specifications*, http://www.unibrain.com/Products/VisionImg/tSpec_Fire_i_BC.htm

[15] Baptista, D. (2007): *Desenvolvimento da Unidade Central de Controlo para uma Plataforma Humanóide*, Relatório Interno, DEMUA-GAR.

[16] PC/104 Consortium (2008): *PC/104 Embedded PC Modules*, http://www.pc104.org/index.php

[17] PC104.com (2008): *What is PC/104?*, http://www.pc104.com/whatis.html

[18] Wikipedia (2007): *Solid-state drive — Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/wiki/Solid-state_drive, [Online; acedido 18-Dez-2007]

[19] iEi Technology Corp (2008): *iEi - PM-LX-800*, http://www.ieiworld.com/index.aspx

[20] Behnke, S.; Stückler, J.; Strasdat H. & Schreiber, M. (2007): *Hierarchical Reactive Control for Soccer Playing Humanoid Robots*, capítulo do livro Humanoid Robots: Human-like Machines, [ISBN: 978-3-902613-07-3], edited by Matthias Hackel, pp. 625-642.

[21] Su, Y.-T. et al.(2007): *Omni-Directional Vision-Based Control Strategy for Humanoid Soccer Robot*, in Proceedings of the 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON), pp. 2950-2955.

[22] E. Menegatti, A. Pretto, A. Scarpa and E. Pagello (2006): *Omnidirectional Vision Scan Matching for Robot Localization in Dynamic Environments*, IEEE Transactions on Robotics, [ISSN: 1552-3098], 22(3):523-535.

[23] Rodrigues, M.; Silva, F. M. & Santos, V. M. (2008): *Visual tracking on an autonomous self-contained humanoid robot*, CLAWAR 2008.

[24] Schulz, H.; Strasdat, H.; & Behnke, S. (2007): *A ball is not just orange: using color and luminance to classify regions of interest*, in Proceedings of the IEEE-RAS International Conference on Humanoid Robots, Nov.29, Pittsburgh, USA.

[25] Pedreiras, P.; Teixeira, F.; Ferreira, N.; Almeida, L.; Pinho, A. & Santos, F. (2006): *Enhancing the reactivity of the vision subsystem in autonomous mobile robots using real-time techniques*, RoboCup Symposium: Papers and Team Description Papers, RoboCup-2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence, Bredenfeld, A.; Jacoff, A.; Noda, I.; Takahashi, Y. (Eds.) pp. 371-383, ISBN: 3-540- 35437-9 , Springer Berlin / Heidelberg , Springer, 2006.

[26] Pedreiras, P.; Almeida, L. (2007): *Task Management for Soft Real-Time Applications Based on General Purpose Operating Systems*, LSE-IEETA/DETI, University of Aveiro, Aveiro, Portugal

[27] Intel Corporation (2008): *Open Source Computer Vision Library*, http://www.intel.com/technology/computing/opencv/index.htm

[28] Asada, M. & Mayer, N. M. (2007): *RoboCup Humanoid Chalenge*, The Second Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Pittsburgh (USA), Nov. 29, 2007

[29] Asada, M.; Kitano, H.; Noda, I. & Veloso, M. (1999): *Robocup: Today and tomorrow - what we have learned. Artificial Intelligence*, 110:193-214, 1999.

[30] Asada, M.; Kitano, H.; Noda, I.; Kuniyoshi, Y.; Osawa, E. & Matsubara, H. (1997): *Robocup: A challenge problem of AI*, AI magazine, 18(1):73-85.

[31] Brooks, R. A. (1991): *Intelligence Without Reason*, A.I. Memo No. 1293, MIT AI Laboratory, April 1991.

[32] Domingos, P. (2007): *Structured Machine Learning: Ten Problems for the Next Ten Years*, Proceedings of Seventeenth International Conference on Inductive Logic Programming, 2007. Corvallis, Oregon: Springer.

[33] Swinson, M. & Bruemmer, D. (2000): *The Expanding Frontiers of Humanoid Robotics*, Intelligent Systems, Julho 2000.

[34] Sanderson, A. C.; Weiss, L. E. & Neuman, C. P. (1987): *Dynamic Sensor-Based Control of Robots with Visual Feedback*, IEEE Transactions on Robotics and Automation, 3:404-417.

[35] Haralick, R. M. & Shapiro, L. G. (1992): *Computer an Robot Vision*, Addison-Wesley Publishing Company, Volume 1

[36] Jähne, B.; Haußecker, H. & Geißler, P. (1999): *Handbook of Computer Vision and Applications Volume I - Sensors and Imaging*, Academic Press, Volume 1

[37] Spong, M. & M. Vidyasagar (1989): *Robot Dynamics and Control*, John Wiley & Sons, Inc. New York, NY, USA, 1989

[38] Behnke, S.; Müller, J. & Schreiber, M. (2006) *Toni: A Soccer Playing Humanoid Robot*, In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, RoboCup-2005: Robot Soccer World Cup IX, pp. 59-70, Lecture Notes in Artificial Intelligence, LNAI 4020, Springer, 2006. Preliminary version in Proceedings of The 9th RoboCup International Symposium, Osaka, Japan, paper #95, July 2005.

[39] Thomas, P.J.; Stonier, R.J. & Wolfs, P.J. (2002): *Robustness of colour detection for robot soccer*, Control, Automation, Robotics and Vision, 2002. ICARCV 2002. 7th International Conference on , vol.3, no., pp. 1245-1249 vol.3, 2-5 Dez. 2002

[40] Hutchinson, S.; Hager, G. & Corke, P. (1996): *A tutorial on visual servo control*, IEEE Transactions on Robotics and Automation 12(5), 651-670.

[41] Hill, J. & Park, W. (1979): *Real time control of a robot with a mobile camera*, in Proceedings of the 9th International Symposium on Industrial Robots, pp. 233-246.

[42] Sanderson, A. & Weiss, L. (1980): *Image-based visual servo control using relational graph error signals*, Proc. IEEE pp. 1074-1077.

[43] Kragic, D. & Christensen, H. I. (2002): *Survey on Visual Servoing for Manipulation*, Tech. Rep. ISRN KTH/NA/P–02/01–SE, Jan. 2002., CVAP259

[44] Corke, P. (1994): *Visual control of robot manipulators - A review*, in K. Hashimoto, ed., 'Visual Servoing', World Scientific, pp. 1-32.

[45] Gonçalves, P. (2005): *Controlo Visual de Robôs Manipuladores*, Dissertação de Doutoramento em Engenharia Mecânica, Instituto Superior Técnico, Abril 2005.

[46] Hashimoto, K. (2003): *A review on vision-based control of robot manipulators*, Advanced Robotics 17 (2003), nº. 10, 969-991.

[47] Hager, G. (1997): *A modular system for robust positioning using feedback from stereo vision*, IEEE Transactions on Robotics and Automation 13(4), 582-595.

[48] Espiau, B.; Chaumette, F. & Rives, P. (1992): *A new approach to visual servoing in robotics*, IEEE Transactions on Robotics and Automation 8(3), 313-326.

[49] Hager, G.; Chang, W. & Morse, A. (1995): *Robot hand-eye coordination based on stereo vision*, IEEE Control Systems Magazine 15(1), 30-39.

[50] Chaumette, F.; Rives, P. & Espiau, B. (1991): *Positioning a robot with respect to an object, tracking it and estimating its velocity by visual servoing*, Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'91' **3**, 2248-2253.

[51] Chaumette, F. (1997): *Potential problems of stability and convergence in image-based and position-based visual servoing*, in 'In Workshop on Vision and Control, Block Island, USA'.

[52] Hashimoto, K. & Noritsugu, T. (1998): *Performance and sensitivity in visual servoing*, in Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'98', Vol. 2, pp. 2321-2326.

[53] Hashimoto, K., Ebine, T. & Kimura, H. (1996): *Visual servoing with hand-eye manipulator-optimal control approach*, IEEE Transactions on Robotics and Automation 12(5), 766-774.

[54] Longuet-Higgins, H. (1981): *A computer algorithm for reconstructing a scene from two projections*, Nature 293, 133-135.

[55] Jerian, C. & Jain, R. (1991): *Structure from motion - A critical analysis of methods*, IEEE Transactions on Systems, Man and Cybernetics 21(3), 572-588.

[56] Malis, E.; Cahumette, F. & Boudet, S. (1998): *Positioning a coarse-calibrated camera with respect to an unknown object by 2D 1/2 visual servoing*, in Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'98', Vol. 1, pp. 1352-1359.

[57] Papanikolopoulos, N.; Khosla, P. & Kanade, T. (1993): *Visual tracking of a moving target by a camera mounted on a robot: a combination of vision and control*, IEEE Transactions on Robotics and Automation 9, 14-35.

[58] Weiss, L.; Sanderson, A. & Neuman, C. (1987): *Dynamic visual servo control of robots: An adaptive image-based approach*, IEEE Journal on Robotics and Automation 3(5), 404-417.

[59] Wikipedia (2008): *Segmentação (processamento de imagem) — Wikipedia, The Free Encyclopedia*, http://pt.wikipedia.org/wiki/Segmentação_(processamento_de_imagem), [Online; acedido 8-Jun-2008]

[60] Sobral, J. L. (2002): *Visão por Computador - Aula 07 Segmentação*, Acetatos das aulas teóricas de 'Visão por Computador', Universidade do Minho, 2002

[61] Lee, D. & Nakamura, Y. (2007): *Motion Capturing from Monocular Vision by Statistical Inference Based on Motion Database: Vector Field Approach*, Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems San Diego, CA, USA, Out 29 - Nov 2, 2007

[62] Chateau, T.; Vacavant, A. & Lavest, J. M. (2005): *Skin Detection and Tracking by Monocular Vision*, Volume 1, 14-15 Julho 2005, pp.79-82 Vol. 1

[63] Kulvanit, P. & Stryk, O. von (2008): *RoboCup Soccer Humanoid League Rules and Setup - draft*, Secção 4.3 - Sensors, Robocup 2008, Suzhou, China, 2008.