

**Faculdade de Engenharia da Universidade do Porto**



## **Sistema de Visão para Aplicações Robóticas**

Rodolfo Nino Mendonça da Silva

VERSÃO FINAL

Dissertação realizada no âmbito do  
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores  
Major Automação

Orientador: Prof. Dr. Hélio Mendes de Sousa Mendonça

Junho 2012



A Dissertação intitulada

“Sistema de Visão para Aplicações Robóticas”

foi aprovada em provas realizadas em 17-07-2012

o júri

  
Presidente Professor Doutor Armando Jorge Miranda de Sousa  
Professor Auxiliar do Departamento de Engenharia Electrotécnica e de  
Computadores da Faculdade de Engenharia da Universidade do Porto

  
Professor Doutor José Luís Magalhães Lima  
Professor Adjunto Departamento de Electrotecnia da Escola Superior de Tecnologia  
e Gestão de Bragança

  
Professor Doutor Hélio Mendes de Sousa Mendonça  
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores  
da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

  
Autor - Rodolfo Nino Mendonça da Silva

Faculdade de Engenharia da Universidade do Porto



## Resumo

Nos últimos anos, tem-se assistido a um crescimento muito acima da média da indústria da visão artificial em comparação com outras indústrias. Inicialmente, os sistemas baseados em visão artificial eram utilizados em áreas muito restritas da indústria. Hoje, estes sistemas são parte integrante de aplicações em muitas áreas.

Os sistemas de visão tradicionais são constituídos por máquinas que têm capacidade de processar em grandes quantidades. Estes sistemas caracterizavam-se também por terem um elevado custo económico. Nas últimas décadas tem-se assistido a uma certa tendência para utilizar a visão artificial em áreas mais vasta uma vez que torna os sistemas mais inteligentes e robustos. Assim, começaram a surgir pequenos dispositivos com uma considerável capacidade de processamento acoplados de uma câmara que permitem realizar tarefas que envolvam visão artificial. Estes sistemas conhecidos como *smart cameras* têm inúmeras vantagens, entre as quais se destaca o seu baixo custo económico.

Neste trabalho será estudada uma destas *smart cameras*, nomeadamente a *Surveyor Blackfin Camera*.

Esta Dissertação no âmbito do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores tem como objetivos estudar a *smart camera* acima referida, criar uma documentação sobre os componentes do sistema e documentar os vários procedimentos para interagir com o mesmo. Pretende-se ainda criar uma aplicação para o sistema que seja capaz de verificar o comportamento do mesmo e permitir identificar os algoritmos possíveis de implementar.

Para a criação da aplicação teve-se por base a secção de Detecção de *Landmarks* de uma dissertação anteriormente desenvolvida pelo estudante Roberto Silva com o tema: “Localização De AGVs Industriais Baseada Em Marcadores” [58].



# Abstract

*In the past years machine vision has enjoyed a growth rate above the average growth of the other industries. Initially systems based on machine vision were deployed in a small range of areas. Nowadays these kinds of systems are an integral part of wide variety of applications in different areas.*

*Traditional systems using machine vision are characterized by having computers with big processing capabilities and for being expensive to afford. In the last few decades there was a trend to use machine vision in simpler applications with less processing loads. This turns in more intelligent and robust systems even in simple applications. Thanks to these new kinds of applications in machine vision, enterprises started to develop micro-processed systems with a coupled camera. These kinds of devices are known as smart-cameras, within many strong points the low cost is remarkable. In this work a smart-camera named Surveyor Blackfin will be studied.*

*This master thesis has the objectives to study the Surveyor Blackfin Camera, produce documentation about the system and the set of procedures to interact with the system. The third objective is to create and implement an algorithm for a selected application. This will allow us to assess the functioning system and to evaluate what kind of algorithms we can implement.*

*To accomplish the final objective the work was based in a former master thesis developed by Roberto Silva with the title: “Localização De AGVs Industriais Baseada Em Marcadores” specifically the section of Landmarks detection [58].*



# Agradecimentos

Agradeço a todos os colegas e professores que me ajudaram ao longo da minha vida académica, em especial aos amigos criados nesta faculdade e ao orientador desta dissertação o Prof. Dr. Hélio Mendes de Sousa Mendonça.



# Índice

|                                                             |           |
|-------------------------------------------------------------|-----------|
| Resumo .....                                                | iii       |
| Abstract .....                                              | v         |
| Agradecimentos .....                                        | vii       |
| Índice .....                                                | ix        |
| Lista de figuras .....                                      | xi        |
| Lista de tabelas .....                                      | xv        |
| Abreviaturas e Símbolos .....                               | xvi       |
| <b>Capítulo 1 .....</b>                                     | <b>1</b>  |
| Introdução .....                                            | 1         |
| 1.1 Enquadramento e Motivação .....                         | 1         |
| 1.2 - Objetivos .....                                       | 2         |
| 1.4 - Estrutura do documento .....                          | 3         |
| <b>Capítulo 2 .....</b>                                     | <b>5</b>  |
| Estado da Arte .....                                        | 5         |
| 2.1 <i>Smart Cameras</i> .....                              | 5         |
| 2.2 Imagem digital .....                                    | 8         |
| 2.3 Captura de imagem .....                                 | 13        |
| 2.4 Processamento Distribuído em Aplicações Robóticas ..... | 15        |
| 2.5 Processamento de imagem .....                           | 16        |
| <b>Capítulo 3 .....</b>                                     | <b>29</b> |
| Sistema em estudo .....                                     | 29        |
| 3.1 Descrição do Hardware .....                             | 30        |
| 3.2 Descrição do <i>Software</i> .....                      | 36        |
| 3.3 Modos de interação com o sistema .....                  | 41        |
| <b>Capítulo 4 .....</b>                                     | <b>45</b> |
| Descrição do sistema AGV .....                              | 45        |
| 4.1 Apresentação geral do AGV .....                         | 45        |
| 4.2 Sistema de visão artificial .....                       | 47        |
| 4.3 Plataformas do <i>Software</i> .....                    | 48        |
| 4.4 Descrição dos <i>Landmarks</i> .....                    | 49        |

|                                                                                                |            |
|------------------------------------------------------------------------------------------------|------------|
| 4.5 Definição dos <i>outputs</i> da <i>smart camera</i> .....                                  | 51         |
| <b>Capítulo 5 .....</b>                                                                        | <b>53</b>  |
| Software desenvolvido.....                                                                     | 53         |
| 5.1 Aplicação de interação com a câmara .....                                                  | 53         |
| 5.2 Processamentos de imagem implementados .....                                               | 56         |
| <b>Capítulo 6 .....</b>                                                                        | <b>71</b>  |
| Resultados .....                                                                               | 71         |
| 6.1 Impacto das melhorias implementadas .....                                                  | 71         |
| 6.2 Tempos de processamento dos algoritmos e suas etapas .....                                 | 73         |
| 6.3 Comparação dos tempos de processamento .....                                               | 77         |
| 6.4 Comparação da robustez dos processamentos .....                                            | 78         |
| <b>Capítulo 7 .....</b>                                                                        | <b>81</b>  |
| Conclusões .....                                                                               | 81         |
| 7.1 Trabalhos futuros .....                                                                    | 82         |
| Anexo A.....                                                                                   | 83         |
| Compilação e Transferência do Firmware em Linux .....                                          | 83         |
| Anexo B.....                                                                                   | 89         |
| Compilação e Transferência do <i>firmware</i> em Windows .....                                 | 89         |
| Anexo C.....                                                                                   | 101        |
| Integração da Blackfin Toolchain no ambiente de desenvolvimento <u>Eclipse</u> - Windows ..... | 101        |
| <b>Bibliografia.....</b>                                                                       | <b>112</b> |

## Lista de figuras

|                                                                                                 |    |
|-------------------------------------------------------------------------------------------------|----|
| Figura 2.1 - Esquema funcional das <i>smart cameras</i> . ....                                  | 5  |
| Figura 2.2 - <i>Surveyor Blackfin Camera</i> . [3] .....                                        | 6  |
| Figura 2.3 - CMUcam3. [4]      Figura 2.4 - AVRcam. [5].....                                    | 7  |
| Figura 2.5 - NTXcam. [7].....                                                                   | 7  |
| Figura 2.6 - NI 1742 smart camera. [9] .....                                                    | 7  |
| Figura 2.7 - A Series Intelligent Camera. [10] .....                                            | 8  |
| Figura 2.8 - Plano de cores U-V para $Y = 0,5$ . [51].....                                      | 10 |
| Figura 2.1 - Posição dos pixéis na trama do formato BMP. ....                                   | 11 |
| Figura 2.2 - Esquema de um sensor CCD. [16] .....                                               | 14 |
| Figura 2.3 - Esquema física dos sensores CMOS. [16].....                                        | 14 |
| Figura 2.4 - Incorporação de circuitos no sensor CMOS. ....                                     | 15 |
| Figura 2.5 - Resultado da aplicação do filtro. ....                                             | 17 |
| Figura 2.6 - Representação das retas em coordenadas polares. [4] .....                          | 21 |
| Figura 2.5 - Ilustração do funcionamento do passo nº 2. [31] .....                              | 23 |
| Figura 2.7 - Esquema ilustrativo da detecção de <i>anchors</i> . ....                           | 25 |
| Figura 2.8 - a) Pesquisa por arestas horizontais; b) Pesquisa por arestas verticais. [31] ..... | 25 |
| Figura 2.9 - Exemplificação do funcionamento do <i>smart routing</i> . [31] .....               | 26 |
| Figura 3.1 - Robô onde se insere a câmara em estudo. [39] .....                                 | 29 |
| Figura 3.2 - Diagrama de blocos funcionais do sensor de imagem. [40] .....                      | 30 |
| Figura 3.3 - Diagrama de blocos do processador. [41] .....                                      | 31 |
| Figura 3.4 - Núcleo do processador. [41] .....                                                  | 34 |
| Figura 3.5 - Esquemático do sistema de geração de <i>reset</i> . [3] .....                      | 35 |

|                                                                                                                  |    |
|------------------------------------------------------------------------------------------------------------------|----|
| Figura 4.1- Robô móvel industrial. [8] .....                                                                     | 45 |
| Figura 4.2 - Diagrama funcional do sistema. [8] .....                                                            | 46 |
| Figura 4.3 - Dimensões e posição da zona de aquisição de imagem. [45] .....                                      | 47 |
| Figura 4. 4 - Câmara DMK 31AU03. [4] .....                                                                       | 47 |
| Figura 4.5 -Interface gráfica da aplicação de análise de imagens. [8].....                                       | 48 |
| Figura 4.6 - Interface gráfica do sistema de controlo. [8] .....                                                 | 48 |
| Figura 4.7 - Interface da aplicação de comunicação com o microcontrolador. [8] .....                             | 49 |
| Figura 4.8 - Forma do marcador desenvolvido. [45].....                                                           | 50 |
| Figura 4.9 - Codificação dos marcadores. [45] .....                                                              | 50 |
| Figura 4.10 - Linha utilizada. [8].....                                                                          | 51 |
| Figura 5.1 - Aspeto da aplicação desenvolvida em Lazarus. ....                                                   | 54 |
| Figura 5.2 - Transformação entre espaços da Transformada de Hough Aleatória. [34] .....                          | 57 |
| Figura 5.3 - Desvio em retas paralelas. ....                                                                     | 67 |
| Figura 5.4 - Representação dos vértices dos marcadores. ....                                                     | 68 |
| Figura 5.5 - Diagrama funcional dos processamentos implementados. ....                                           | 69 |
| Figura 6.1 - Resultado da parametrização com o uso de apenas 2 pontos. ....                                      | 73 |
| Figura 6. 2 - Distribuição dos tempos de processamento das várias etapas. Tempo em ms. ..                        | 74 |
| Figura 6. 3 -Distribuição dos tempos de processamento das várias etapas EDlines. Tempo em ms. ....               | 76 |
| Figura 6.4 - Imagem utilizada à esquerda, processamento RHT ao centro e EDlines à direita .....                  | 78 |
| Figura 6.5 - Imagem utilizada à esquerda, processamento RHT ao centro e EDlines à direita. ....                  | 78 |
| Figura 6.6 - Imagem utilizada à esquerda, processamento RHT ao centro e EDlines à direita. ....                  | 78 |
| Figura 6. 7 - Imagem com densidade de ruído uniforme de 0,03. Resultado do processamento Edlines à direita. .... | 79 |
| Figura 6. 8 - Imagem com densidade de ruído uniforme de 0,05. Resultado do processamento Edlines à direita. .... | 79 |
| Figura A.1 - Mensagem recebida após compilação.....                                                              | 84 |
| Figura A.2 - Aspeto do Moserial. ....                                                                            | 85 |
| Figura A.3 - Configurações. ....                                                                                 | 86 |
| Figura A.4 - Configuração da porta série. ....                                                                   | 86 |

|                                                        |     |
|--------------------------------------------------------|-----|
| Figura A.5 - Versão do firmware. ....                  | 87  |
| Figura A.6 - Envio do arquivo. ....                    | 87  |
| Figura A.7 - Seleção do arquivo. ....                  | 88  |
| Figura B.1 - Instalação Mingw32 1. ....                | 90  |
| Figura B.2 - Instalação Mingw32 2. ....                | 90  |
| Figura B.3 - Instalação Mingw32 3. ....                | 91  |
| Figura B.4 - Instalação Mingw32 4. ....                | 91  |
| Figura B.5 - Instalação Mingw32 5. ....                | 92  |
| Figura B.6 - Instalação Mingw32 6. ....                | 92  |
| Figura B.7 - Instalação <i>Toolchain</i> 1. ....       | 93  |
| Figura B.8 - Instalação <i>Toolchain</i> 2. ....       | 93  |
| Figura B.9 - Instalação <i>Toolchain</i> 3. ....       | 94  |
| Figura B.10 - Instalação <i>Toolchain</i> 4. ....      | 94  |
| Figura B.11 - Instalação <i>Toolchain</i> 5. ....      | 95  |
| Figura B.12 - Propriedades do sistema. ....            | 96  |
| Figura B.13 - Variáveis de ambiente. ....              | 96  |
| Figura B.14 - Edição de variável do sistema. ....      | 97  |
| Figura B.15 - Compilação 1. ....                       | 97  |
| Figura B.16 - Compilação 2. ....                       | 98  |
| Figura B.17 - Compilação 3. ....                       | 98  |
| Figura B.18 - Configuração da porta série. ....        | 99  |
| Figura B.19 - Envio do ficheiro loader. ....           | 99  |
| Figura B.20 - Seleção do ficheiro. ....                | 100 |
| Figura C.1 - Ativação do site. ....                    | 102 |
| Figura C.2 - Instalação CDT. ....                      | 103 |
| Figura C.3 - Instalação da <i>Toolchain</i> . ....     | 104 |
| Figura C.4 - Instalação Zylind CDT. ....               | 105 |
| Figura C.5 - Instalação do Terminal. ....              | 105 |
| Figura C.6 - Seleção do Terminal para instalação. .... | 106 |
| Figura C.7 - Conclusão da instalação do terminal. .... | 107 |

|                                                 |     |
|-------------------------------------------------|-----|
| Figura C.8 - Criação de um novo projeto.....    | 108 |
| Figura C.9 - Seleção da <i>Toolchain</i> . .... | 109 |
| Figura C.10 - Seleção do processador. ....      | 110 |
| Figura C.11 - Compilação concluída. ....        | 111 |

## Lista de tabelas

|                                                                                       |    |
|---------------------------------------------------------------------------------------|----|
| Tabela 2.1 - Comparação de algumas <i>smart cameras</i> existentes no mercado. ....   | 6  |
| Tabela 2.2 - Bitmap File Header .....                                                 | 12 |
| Tabela 2.3 - Bitmap info.....                                                         | 12 |
| Tabela 2.4 - Exemplo de acumulador para a detecção de retas. ....                     | 20 |
| Tabela 3. 1 - Principais zonas da memória para as imagens. ....                       | 38 |
| Tabela 4. 1 - Formato da trama resultante do processamento de imagem. [45] .....      | 49 |
| Tabela 4.2 - Dimensões do marcador. [45].....                                         | 50 |
| Tabela 6.1 - Tabela resumo das melhorias implementadas na transformada de Hough. .... | 72 |
| Tabela 6.2 - Tabela resumo das melhorias implementadas no EDlines. ....               | 73 |
| Tabela 6. 3 - Tempos de processamento nas várias resoluções. ....                     | 74 |
| Tabela 6.4 - Tempos de processamento nas várias resoluções. ....                      | 76 |
| Tabela 6.5 - Resumos dos tempos de processamento totais. ....                         | 77 |

# Abreviaturas e Símbolos

## Lista de Abreviaturas

|         |                                                |
|---------|------------------------------------------------|
| ALU     | <i>Arithmetic logic unit</i>                   |
| BMP     | <i>Bitmap Image File</i>                       |
| CAN     | <i>Controller Area Network</i>                 |
| CC      | <i>Corrente contínua</i>                       |
| CCD     | <i>Charge-coupled device</i>                   |
| cm      | <i>Centímetro</i>                              |
| CMOS    | <i>Complementary metal-oxide-semiconductor</i> |
| EDLINES | <i>Edge Drawing Line Segmentation</i>          |
| FIFO    | <i>First In First Out</i>                      |
| FPS     | <i>Frames Per Second</i>                       |
| GHz     | <i>gigahertz</i>                               |
| GIF     | <i>Graphics Interchange Format</i>             |
| HT      | <i>Hough Transform</i>                         |
| HTTP    | <i>Hypertext Transfer Protocol</i>             |
| I2C     | <i>Inter-Integrated Circuit</i>                |
| JPEG    | <i>Joint Photographic Experts Group</i>        |
| JTAG    | <i>Joint Test Action Group</i>                 |
| MB      | <i>MegaByte</i>                                |
| MHZ     | <i>MegaHertz</i>                               |
| MISO    | <i>Master In Slave Out</i>                     |
| MOSI    | <i>Master Out Slave In</i>                     |
| mW      | <i>miliwatt</i>                                |
| °C      | <i>Grau centígrado</i>                         |
| PNG     | <i>Portable Network Graphics</i>               |
| PPI     | <i>Parallel Peripheral Interface</i>           |
| PWM     | <i>Pulse-Width Modulation</i>                  |
| RAM     | <i>Random Access Memory</i>                    |
| RGB     | <i>Red, Green, Blue</i>                        |
| RHT     | <i>Randomized Hough Transform</i>              |
| RISC    | <i>Reduced Instruction Set Computer</i>        |
| RS232   | <i>Serial Protocol</i>                         |
| RTC     | <i>Real Time Clock</i>                         |

|       |                                                    |
|-------|----------------------------------------------------|
| SCCB  | <i>Serial Camera Control Bus</i>                   |
| SCK   | <i>Serial Clock</i>                                |
| SCL   | <i>Signal Clock</i>                                |
| SDA   | <i>Signal Data</i>                                 |
| SDRAM | <i>Synchronous dynamic random-access memory</i>    |
| SIMD  | <i>Single Instruction Multiple Data</i>            |
| SPI   | <i>Serial Peripheral Interface</i>                 |
| SPORT | <i>Full-Duplex Synchronous Serial Ports</i>        |
| TIFF  | <i>Tagged Image File Format</i>                    |
| TWI   | <i>Two Wire Interface</i>                          |
| UART  | <i>Universal Asynchronous Receiver/Transmitter</i> |
| USB   | <i>Universal Serial Bus</i>                        |
| uW    | <i>microwatt</i>                                   |
| V     | <i>Volt</i>                                        |
| VGA   | <i>Video Graphics Array</i>                        |
| W     | <i>Watt</i>                                        |



# Capítulo 1

## Introdução

Neste capítulo é feita uma descrição do âmbito deste projeto, dos seus objetivos, da metodologia adotada e ainda da estrutura deste documento.

### 1.1 Enquadramento e Motivação

A indústria da visão artificial está cada vez mais presente no quotidiano da sociedade moderna. Inicialmente era utilizada apenas em áreas e aplicações específicas, no entanto, com o passar dos tempos essa realidade foi-se modificando.

Com a grande capacidade de processamento dos novos computadores, os campos de aplicação da visão artificial têm vindo a crescer cada vez mais. Com a possibilidade de fazer grandes quantidades de processamento em imagens, novas portas se abriram à criação de algoritmos mais complexos e robustos que podem ser aplicados a vastas áreas tais como: linhas de assemblagem, controlo de qualidade, diagnóstico de patologias, aplicações militares e muito recentemente em veículos autónomos.

Se por um lado, sistemas baseados em visão são utilizados para aplicações muito complexas, por outro lado, cada vez mais se tira partido desta tecnologia em aplicações mais simples sem necessidade de algoritmos muito complexos que acabam por ser muito “pesados” em termos de processamento.

Para que se possa utilizar a visão artificial em aplicações mais simples é necessário haver sistemas que nos permitam a captura, processamento e extração de características a um custo não muito elevado. Inicialmente, os sistemas baseados em visão eram muito dispendiosos. A utilização da visão artificial em aplicações mais simples exige que os sistemas sejam economicamente mais acessíveis.

Devido à necessidade anteriormente explicada começaram a aparecer no mercado sistemas integrados com câmaras e microprocessadores acoplados, que permitem capturar imagens e implementar algoritmos para a extração de dados.

Estes sistemas denominados de *smart cameras* com os seus relativos baixos custos e boas capacidades de processamento permitem utilizar a visão artificial em aplicações de baixa complexidade, obtendo assim aplicações mais inteligentes e fiáveis. Historicamente, o aumento da capacidade dos novos computadores abriu portas à criação de algoritmos mais complexos. Atualmente existe a tendência para melhorar a eficiência dos algoritmos já existentes. Para além disso, procura-se criar novos e mais eficientes algoritmos que produzam os mesmos resultados que os anteriores.

Com a crescente necessidade de utilizar e consequentemente estudar as *Smart Cameras* surgiu este trabalho no âmbito da unidade curricular de Dissertação do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores.

Neste trabalho irá ser estudada uma *Smart Camera* denominada *Surveyor Blackfin Camera*.

## 1.2 - Objetivos

Esta dissertação de mestrado tem os seguintes principais objetivos:

- Estudar a Surveyor Blackfin Camera.
- Criar uma documentação sobre os componentes do sistema e os vários procedimentos para interagir com o mesmo e programa-lo. Pretende-se produzir um documento que possa ser consultado por todos os interessados em trabalhar com a *smart camera* e desta forma possam rapidamente adquirir todo o *know-how* necessário para poder utilizar o sistema.
- Por fim, criar uma aplicação que permita verificar o sistema a funcionar e ter uma ideia dos algoritmos possíveis de implementar no sistema. Para a criação desta aplicação irá ser utilizado um trabalho anteriormente desenvolvido no âmbito da Dissertação do MIEEC pelo estudante David Lima e posteriormente melhorado também no âmbito da Dissertação do MIEEC pelo estudante Roberto Silva, em especial a secção de Detecção de *Landmarks*. Trata-se de um sistema de localização de AGVs com base no reconhecimento de imagem. O sistema foi desenhado da seguinte forma: a câmara existente no AGV captura as imagens do percurso e envia-a para o computador. O processamento da imagem e o algoritmo da localização é executado no computador. Mais detalhes sobre este sistema serão fornecidos no capítulo 4 deste documento. Pretende-se passar esse processamento de imagem e o algoritmo de imagem para o nosso sistema. Neste trabalho pretende-se realizar esse processamento *on-board* da *Surveyor Blackfin Camera*. Pretende-se produzir um algoritmo eficiente e robusto tendo sempre em conta a capacidade de processamento do sistema.

Decidiu-se primeiramente fazer um estudo ao sistema. Começou-se por estudar cada componente físico do sistema e depois estudou-se o sistema numa perspetiva de

*software*. Criou-se vários tutoriais sobre como interagir com o sistema e como programa-lo.

Posto isto iniciou-se a fase de alteração do *firmware* disponibilizado pelo fabricante. Inicialmente com alterações menos significativas e depois com alterações maiores. A partir do momento em que o processo de alteração estava já bem estudado e com a investigação anterior feita a nível de *software* realizou-se uma “limpeza” ao *firmware* original de modo a reduzir substancialmente a complexidade do *software*.

Passou-se a uma nova fase do trabalho de maneira a cumprir o último objetivo descrito na secção 1.2. . Estudou-se o sistema utilizado pelo estudante Roberto Silva e fez-se uma pesquisa bibliográfica de algoritmos possíveis de ser utilizados nesta aplicação. De seguida implementou-se os algoritmos selecionados. Por fim criou-se este documento que contém uma prévia pesquisa bibliográfica de assuntos de interesse para esta dissertação e irá descrever todos os procedimentos efetuados, algoritmos implementados e resultados obtidos.

## 1.4 - Estrutura do documento

Este documento possui 7 capítulos que introduzem o leitor ao domínio do trabalho, apresentam as referências bibliográficas relevantes para o trabalho a desenvolver, descrevem os métodos e ferramentas utilizados e por fim mostram os resultados obtidos bem como uma análise aos mesmos.

O Capítulo 1 trata-se de uma introdução ao trabalho e uma apresentação dos objetivos, problemas a tratar, metodologias e da estrutura deste documento.

No Capítulo 2 são expostos diversos textos de temas com interesse para o desenvolvimento deste trabalho. Serão abordados os seguintes temas: equipamentos similares à *smart camera* em estudo, os vários tipos de formatos de imagem e formatos de compressão, a captura física da imagem, algoritmos de processamento de imagens e das vantagens do processamento em sistemas robóticos ser distribuído. Estes textos produzidos são resultado da pesquisa bibliográfica efetuada no âmbito deste projeto.

A descrição do sistema em estudo encontra-se no Capítulo 3. Aqui é feita uma descrição do *hardware* da plataforma *Surveyor Blackfin Camera* e também da arquitetura e das funcionalidades do *firmware* disponibilizado pelo fabricante.

No Capítulo 4 é feita uma apresentação dos *Landmarks* a serem detetados e do sistema onde essa deteção se insere. São também definidos os *outputs* dos algoritmos de processamento de imagem a serem implementados.

O Capítulo 5 centra-se no *software* desenvolvido neste projeto (tanto o desenvolvido no computador para interagir com o sistema como o desenvolvido para ser processado *on-board* da plataforma).

Todos os resultados da implementação dos algoritmos de processamento das imagens, comparação e análise dos mesmos encontram-se no Capítulo 6.

Por fim é feita uma conclusão acerca de todo o trabalho desenvolvido e dos seus resultados.



# Capítulo 2

## Estado da Arte

### 2.1 *Smart Cameras*

#### 2.1.1 Introdução

As *smart cameras* [1] são dispositivos capazes de não só adquirir imagens do mundo real mas também entendê-las. Ron Schneidermann foi o primeiro autor a definir o termo *smart camera* em 1975 como uma câmara capaz de realizar uma técnica usada pelos fabricantes de câmaras fotográficas para controlar a exposição à luz através do controlo da velocidade de disparo. Em 1981, Richard Lyon inventou o primeiro rato de computador óptico e assim produziu-se um dispositivo com uma câmara para capturar imagem e uma unidade de processamento embebida. Para além do rato de computador óptico a visão artificial também teve um papel fundamental na evolução destes tipos de câmaras. A partir dos anos 80 muitas empresas investiram nesta área das câmaras inteligentes de baixo custo e muitas delas ainda hoje em dia continuam a ser casos de sucesso.

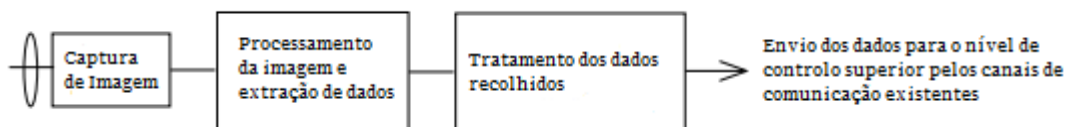


Figura 2.1 - Esquema funcional das *smart cameras*.

No geral as *smart cameras* [2] são compostas por um sensor de imagem (a câmara propriamente dita), uma unidade de processamento para interpretar as imagens vindas do sensor, interfaces para comunicação (*USB*, *RS232*, *Ethernet*, *Wi-fi*, *JTAG*, etc). São normalmente dispositivos compactos.

Geralmente, as câmaras já trazem algumas funções de visão artificial *standard*, mas comumente estas funções são alteradas para aplicações específicas. A utilização

de pequenos CPU's acoplados à câmara veio também baixar a carga de processamento do CPU central em sistemas de processamento distribuído.

Hoje em dia existem *smart cameras* que podem rivalizar com computadores em termos de capacidade de processamento. No entanto no âmbito desta dissertação serão apenas estudadas as câmaras com capacidades de processamento mais baixas e consequentemente mais baratas sendo por isso designadas de “*low cost*”.

Na secção seguinte irão ser apresentadas algumas das Smart Cameras existentes no mercado e feita uma comparação entre elas.

### 2.1.2 Equipamentos existentes no mercado

As especificações relativas a cada câmara em análise encontram-se apresentadas na tabela seguinte:

**Tabela 2.1 - Comparação de algumas *smart cameras* existentes no mercado.**

|                           | Processador (MHz) | Memória | Comunicação                 | Res. Máx. | Fps Máx. | Consumo Máx.(W) | Preço (€) |
|---------------------------|-------------------|---------|-----------------------------|-----------|----------|-----------------|-----------|
| Surveyor Blackfin Cam.    | 500               | 32+4 MB | SPI,UART,I2C, JTAG,Ethernet | 640x480   | 30       | 0,479           | 130       |
| CMUcam3                   | 60                | 128 KB  | UART                        | 640x480   | 8        | 0,300           | 100       |
| NI 1742 smart cam.        | 400               | 128MB   | Ethernet, RS232             | 640x480   | 60       | 0,200           | 1400      |
| AVR cam                   | 16                | 8KB     | RS232, I2C                  | 88x144    | 30       | 0,280           | 65        |
| A Series Intelligent cam. | 533               | 1024    | RS232, Ethernet, LAN/WAN    | 752x480   | 69       | 1               | n/d       |
| NTX CAM v4                | 16                | 8+1 KB  | I2C,USB                     | 80x144    | 30       | 0,197           | 100       |

### 2.1.3 - Comparação e análise

Antes de mais importa realçar que não existem soluções ótimas para todos os tipos de aplicações.



**Figura 2.2 - Surveyor Blackfin Camera. [3]**

Começando pela Surveyor Blackfin Camera, esta é uma plataforma *open source* que se destaca pela sua relativa grande capacidade de processamento quando comparada a câmaras com preços similares. Destaca-se também o fato de ter uma grande comunidade no fórum

disponibilizado pelo fabricante para troca de informações entre os utilizadores. Um aspeto negativo é o facto de que as informações de como interagir com a câmara, configurações, etc encontram-se muitas dispersas pelo fórum.



Figura 2.3 - CMUcam3. [4]



Figura 2.4 - AVRcam. [5]

A CMUcam 3 [3] é, em comparação às outras apresentadas a que está no mercado há mais tempo. Toda a documentação relativa à câmara e tutoriais relativos à operação com esta encontram-se disponíveis no sítio da internet do fabricante. Destaca-se o grande suporte por parte do fabricante como ponto positivo e como ponto negativo o facto de o *firmware* disponibilizado por aquele conter alguns *bugs*. A AVRcam é muito similar à câmara anteriormente apresentada mas tem uma comunidade de desenvolvimento muito menor quando comparada com a CMUcam3. Também a experiência dos vários utilizadores deste sistema diz que o suporte por parte do fabricante é muito fraco.



Figura 2.5 - NTXcam. [7]

A NTXcam é uma câmara que ao contrário das anteriormente apresentadas não é *open source*. É muito semelhante à AVRcam e começa a existir alguma documentação e código *open source* relativos a esta câmara produzida pelos utilizadores. [4]



Figura 2.6 - NI 1742 smart camera. [9]

NI 1742 é uma câmara construída a pensar nas interferências eletromagnéticas existentes em ambientes industriais. Possui cabos de comunicação imunes ao ruído e em alguns *inputs* e *outputs* existe acoplamento óptico para proteção dos circuitos internos. Para além disso evidencia-se a facilidade de produzir programas para esta câmara utilizando ferramentas desenvolvidas pelo mesmo fabricante (*National Instruments*) como por exemplo o LABVIEW. A sua grande desvantagem é o seu elevado custo.



**Figura 2.7 - A Series Intelligent Camera. [10]**

A *Series Intelligent camera* é um equipamento altamente preparado para ambientes industriais. Possui um suporte físico muito robusto a impactos pelo que pode ser utilizado em sistemas robóticos altamente móveis. A informação disponibilizada pelo fabricante é muito restrita pelo que nos impede de tirar maiores conclusões.

Em suma pode-se dizer que em termos da relação performance/preço a *Surveyor Blackfin Camera* é a que apresenta melhor rácio.

## 2.2 Imagem digital

O termo imagem origina da expressão latina *imago* que significa figura, sombra e imitação. [5] diz que: "... a imagem é tida como representação inteligível de alguns objetos com capacidade de ser reconhecida pelo homem necessitando de concretizar-se materialmente ...".

Até chegarmos à imagem digital, vários autores, ao longo de vários séculos realizaram diversos desenvolvimentos na área da fotografia analógica. Entre vários acontecimentos destaca-se a invenção da câmara escura cuja origem se desconhece, utilizada na época do renascimento por artistas para fazer esboços de pinturas. No entanto, esta não permitia guardar a imagem para ser visualizada num momento posterior. Mais tarde com a descoberta de materiais fotossensíveis tornou-se possível registar imagens. No início os materiais utilizados faziam com que o tempo de exposição luz fosse muito elevado (várias horas). Com o passar dos tempos novos materiais e novas técnicas fizeram com que o período de exposição fosse muito baixo e se pudesse guardar as imagens em negativos.

A primeira imagem digital foi criada por Russell Kirsch [6] em 1957 que utilizou um dispositivo (protótipo dos modernos *scanners*) que convertia uma imagem analógica numa matriz binária.

Existem principalmente dois tipos de imagem digitais, as *raster* e as vetoriais. As imagens *raster* contêm informação acerca da cor de cada pixel. As imagens vetoriais são imagens que contêm formas geométricas tais como elipses, polígonos, curvas que podem ser descritas por vetores matemáticos. Estes tipos de imagens podem ser redimensionáveis sem perda de qualidade de imagem. As imagens vetoriais podem ser representadas por parâmetros matemáticos que descrevem as curvas existentes na figura e como consequência estas imagens são geralmente mais pequenas (em número de bits) que as imagens raster.

Para uma explicação mais simples da estrutura das imagens digitais *raster* considera-se primeiramente as imagens monocromáticas, isto é, apenas com tons de cinzentos e só posteriormente irá ser abordada a questão de adição de cor.

Uma imagem digital monocromática [7] é uma função bidimensional  $f(x,y)$  (em coordenadas cartesianas) da intensidade luminosa nos vários pontos da imagem. Sendo a luz uma forma de energia a função acima referida está compreendida entre os seguintes valores:  $0 \leq f(x,y) \leq +\infty$ . Esta é uma função contínua que precisa ser amostrada espacialmente e passar por um processo de quantização dos valores da luz para se poder produzir uma imagem digital. A amostragem espacial é feita ao longo de diferentes tipos de grelhas (matrizes) nomeadamente: retangulares, hexagonais, quadradas. A título de exemplo considera-se uma grelha retangular. O tamanho das grelhas é dado pela resolução das imagens.

$$\begin{bmatrix} f(0,0) & f(1,0) & f(2,0) & \dots & f(M-1,0) \\ f(0,1) & \dots & \dots & \dots & f(M-1,1) \\ \dots & \dots & \dots & \dots & \dots \\ f(0,N-1) & f(1,N-1) & \dots & \dots & f(M-1,N-1) \end{bmatrix}$$

Em que M e N são o número de píxeis na horizontal e vertical respetivamente.

Para explicar a adição de cores à imagem digital é necessário uma breve explicação sobre o olho humano. O olho humano [8] contém predominantemente três recetores que são sensíveis a três áreas do espectro visível cujos picos de sensibilidade se encontram aos 445 nanómetros (cor azul), 535 nanómetros (cor verde) e 570 nanómetros (cor vermelha). Isto faz com que qualquer uma cor possa ser representada para o olho humano como uma mistura de três cores apropriadamente escolhidas (cores primárias). Normalmente as cores primárias são vermelho, verde e azul ou vermelho verde e violeta.

### 2.2.1 Espaços de cores

Nesta subsecção pretende-se apresentar ao leitor uma descrição sobre dois espaços de cores selecionados: RGB e YUV. O espaço de cores RGB é o utilizado nos formatos de ficheiros de imagem BMP. Este tipo de ficheiro será utilizado no desenvolvimento de rotinas de processamento de imagem. Desta forma consegue-se obter imagens sem qualquer tipo de compressão (sem perda de informação). O espaço de cores YUV é o utilizado pela *smart camera* em estudo, ou seja, na memória RAM do sistema está representada a imagem captura no espaço de cores YUV.

#### 2.2.1.1 RGB

O RGB [9] é um espaço de cores, define três cores primárias que descrevem a coloração de cada pixel da imagem. O nome RGB deriva das iniciais das cores definidas: vermelhas (*Red*), verde (*Green*) e azul (*Blue*).

Estas cores foram definidas desta maneira devido ao facto de que as suas frequências serem mais facilmente sensibilizadas pelo olho humano como mencionado anteriormente.

Em termos computacionais cada pixel corresponde a 3 *bytes*, cada *byte* contendo o valor das diferentes componentes. Uma das principais desvantagens deste espaço de cores é o facto de a componente de brilho não estar embutida diretamente no pixel. Assim se for pretendido alterar o brilho da imagem é necessário efetuar cálculos computacionais pesados para cada pixel.

### 2.2.1.2 YUV

Historicamente o espaço de cores YUV [10] foi desenvolvido para haver compatibilidade entre sistemas de televisão analógica a preto e branco e os sistemas a cores. Y definido como luminância representa a componente de brilho. U e V determinam a cor. No caso digital as tanto Y,U e V variam numa gama de 0 a 255. Geralmente os valores de YUV são obtidos através dos valores de RGB através das seguintes equações:

$$Y = 0,299R + 0,587G + 0,114B \quad (2.1)$$

$$U = -0,147R - 0,289G + 0,436B \quad (2.2)$$

$$V = 0,615R - 0,515G - 0,100B \quad (2.3)$$

Uma das vantagens deste espaço de cores é o facto de se poder isolar facilmente a componente de luminância (Y) para posterior processamento de imagem. No entanto as componentes que adicionam cor à imagem podem ser úteis nalguns tipos de processamento como na Estimativa de Movimento.

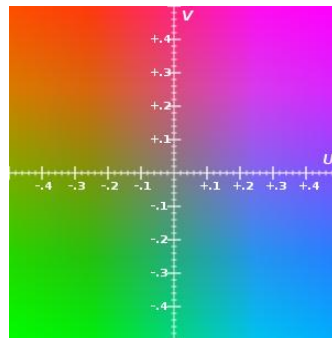


Figura 2.8 - Plano de cores U-V para Y = 0,5. [51]

### 2.2.2 Formatos de Imagem

Com o decorrer dos tempos vários formatos de imagem foram criados, uns foram desaparecendo e outros continuaram a ser usados. Os formatos mais simples limitam-se a indicar a largura e altura da imagem, o tipo de imagem (se em cores ou em tons de cinzento) e os valores RGB de cada pixel.

Alguns dos formatos mais comumente utilizados são os seguintes:

- **BMP** [11] - Criado pela *Microsoft* e é suportado por todas as variantes dos sistemas operativos Windows.
- **GIF** [12] - Foi desenvolvido pela CompuServe Inc. e foi criado para transmissões de imagens em tecnologias de comunicação de baixa largura de banda.
- **PNG** [13] - Surgiu para contornar as limitações técnicas e restrições legais resultantes dos direitos de propriedade do formato GIF.

- **TIFF** [14] - É um formato muito utilizado em aplicações profissionais nomeadamente em aplicações médicas e industriais. Este tipo de ficheiro foi criado pela empresa *Aldus* (mais tarde adquirida pela Adobe) e é muito utilizado em digitalizações.
- **JPEG** [13] - Resulta da norma internacional JPEG (ISO 10918-1). Seu nome deriva das iniciais de *Joint Photographic Experts Group*.

Agora, com mais detalhe descreve-se dois formatos (BMP e JPEG) que terão particular interesse no âmbito desta dissertação. Existe a necessidade de conhecer mais a fundo o formato BMP visto que se pretende construir uma rotina que envie as imagens em formato BMP para o computador de maneira a não haver perda de informação.

### 2.2.2.1 Formato BMP

O formato BMP (também conhecido como DIB) foi criado pela *Microsoft* e é suportado em todas variantes do sistema operativo Windows. Neste formato as imagens são descritas por varrimento ascendente das linhas, ou seja, o ficheiro contém primeiramente a informação dos pixéis referentes à última linha da imagem, depois à penúltima, depois à antepenúltima e assim sucessivamente até chegar à primeira linha. Relativamente aos pixéis de cada linha estes são varridos da esquerda para a direita. Na figura seguinte apresenta-se uma ilustração da posição dos pixéis na trama da imagem.

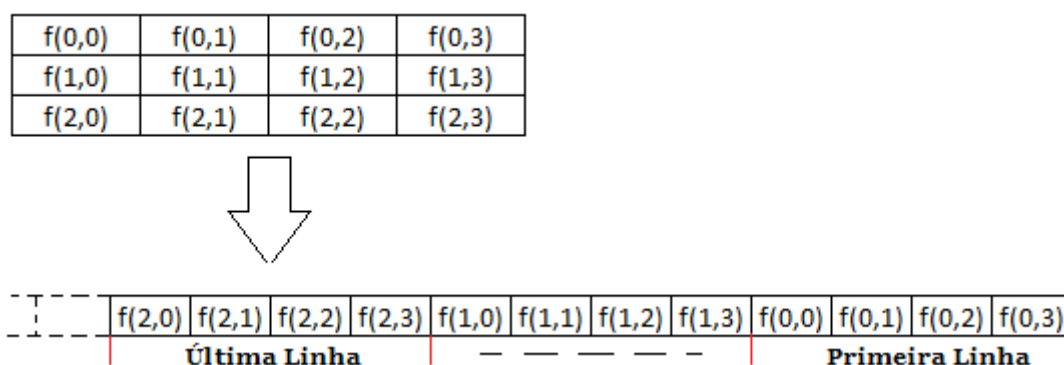


**Figura 2.1 - Posição dos pixéis na trama do formato BMP.**

A informação contida no ficheiro do tipo BMP encontra-se dividida em três blocos principais:

1. **Bitmap File Header** - Contém informações acerca do ficheiro.
2. **Bitmap Info** - Contém informações acerca da imagem.
3. **Dados da imagem** - Contém os valores RGB dos vários pixéis que compõem a imagem.

Os valores inteiros encontram-se armazenados com os bytes menos significativos em primeiro lugar.

Este formato de imagem não sofre de perda de informação devido à não existência de métodos de compressão. No entanto origina imagens muito grandes.

De seguida encontram-se a descrição dos bytes que formam os *headers* [11] do formato BMP.

**Tabela 2.1 - Bitmap File Header**

| Start byte | Tamanho (nº bytes) | Designação  | Valor Por defeito | Descrição                                                     |
|------------|--------------------|-------------|-------------------|---------------------------------------------------------------|
| 1          | 2                  | bfType      | 19778             | Corresponde a 'BM', que identifica o tipo de ficheiro.        |
| 3          | 4                  | bfSize      |                   | Especifica o tamanho do ficheiro em bytes.                    |
| 7          | 2                  | bfReserved1 | 0                 | Deve estar sempre a zero.                                     |
| 9          | 2                  | bfReserved2 | 0                 | Deve estar sempre a zero.                                     |
| 11         | 4                  | bfOffBits   | 1078              | Indica a partir de que byte começam os valores RGB da imagem. |

**Tabela 2.2 - Bitmap info.**

| Start byte | Tamanho (nº bytes) | Designação      | Valor Por defeito | Descrição                                                                            |
|------------|--------------------|-----------------|-------------------|--------------------------------------------------------------------------------------|
| 15         | 4                  | biSize          | 40                | Nº de bytes do <i>Info Header</i>                                                    |
| 19         | 4                  | biWidth         | 100               | Nº de pixéis na horizontal                                                           |
| 23         | 4                  | biHeight        | 100               | Nº de pixéis na vertical                                                             |
| 27         | 2                  | biPlanes        | 1                 | Nº de planos do dispositivo de display                                               |
| 29         | 2                  | biBitCount      | 24                | Nº de bits por pixel                                                                 |
| 31         | 4                  | biCompression   | 0                 | Método de compressão                                                                 |
| 35         | 4                  | biSizeImage     | 0                 | Senão houver compressão este valor deverá ser nulo                                   |
| 39         | 4                  | biXpelsPerMeter | 0                 | Nº de pixéis por metro no dispositivo de display. Pode ser valor nulo.               |
| 43         | 4                  | biYpelsPerMeter | 0                 | Nº de pixéis por metro no dispositivo de display. Pode ser valor nulo.               |
| 47         | 4                  | biClrUsed       | 0                 | Nº de cores do bitmap, se tiver em 0 o valor é calculado através do campo biBitCount |
| 51         | 4                  | biClrImportant  | 0                 | Nº de cores importantes para a imagem, se tiver a 0 todas as cores são importantes   |

#### **2.2.2.2 Formato JPEG**

Resulta da norma internacional JPEG (ISO 10918-1). Seu nome deriva das iniciais de *Joint Photographic Experts Group*. Esta norma define um conjunto de algoritmos de compressão e descompressão com e sem perda de informação. É normalmente utilizado em imagens fotográficas e através de uma perda de qualidade controlada das mesmas consegue reduzir o seu tamanho. Uma vez que a visão humana não consegue verificar as diferenças muito diminutas de cor estes algoritmos conseguem reduzir em grande escala o tamanho das fotos sem que o utilizador consiga notar à primeira vista.

É um formato muito popular graças à expansão da internet em que os utilizadores precisavam de transmitir imagens com boa qualidade mas com baixo número de bytes.

Este formato utiliza o espaço de cores YUV (luminância, cromaticidade azul e cromaticidade vermelha).

A compressão JPEG [13] utiliza algoritmos de compressão e descompressão do tipo da transformada discreta co-seno aplicados a grupos 8x8 de píxeis da mesma cor. Como resultado dos algoritmos temos os coeficientes AC e DC que depois de sofrerem tratamento são armazenados nos blocos de informação dos ficheiros. Estes são constituídos por blocos separados de informação separados por marcas com o comprimento de um byte. Todos os eles começam com o segmento SOI que assinala o início da imagem e é seguido por um segmento do tipo APP0 que identifica o tipo de JPEG, o ficheiro poderá posteriormente conter vários ficheiros APPn em que n significa o número de ordem. Depois existem: os segmentos DQT (que contêm as tabelas de quantização), os segmentos SOF0 (que descrevem os principais parâmetros da imagem), os blocos DHT (que têm os valores das tabelas de Huffman utilizados na compressão e são necessários para a descompressão), o bloco SOS (que descreve as zonas da imagem comprimidas). O bloco EOI assinala o fim do ficheiro.

### **2.3 Captura de imagem**

A função principal de uma câmara é a captura de uma imagem da realidade envolvente, pelo que o componente mais importante da mesma é o sensor digital. Existem dois principais tipos de tecnologia destes sensores: os CCD (*Charged-Coupled Device*) e os CMOS (*Complementary Metal-Oxide-Semiconductor*).

#### **2.3.1 Sensores CCD**

Os sensores CCD [15] foram criados em 1969 nos laboratórios Bell por Williard Boyle e George Smith. Estes sensores recebem a luz nos seus elementos foto sensores (fotodíodos) através do efeito fotoelétrico.

Na figura seguinte observa-se o esquema de um sensor CCD bidimensional.

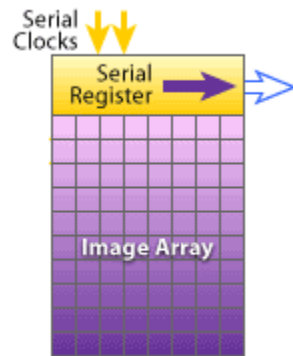


Figura 2.2 - Esquema de um sensor CCD. [16]

A leitura da imagem digital [15] é feita através da transferência das cargas elétricas armazenadas em cada pixel para a linha de leitura do registro (*Serial Readout Register*) e seguem para um conversor analógico-digital. Os valores da tensão podem ser posteriormente amplificados. De reparar que as linhas de pixéis mais “afastadas” do *Serial Readout Register* demoraram mais tempo a ser lidas devido aos sucessivos *shifts* das linhas e consequentemente ficam mais tempo expostas à luz. Para evitar esta situação é utilizado um sistema mecânico que evita a exposição dos fotodíodos à luz.

Os CCD [17] apresentam uma boa imunidade ao ruído, possuem uma maior sensibilidade relativamente aos CMOS e como são fabricados há muito mais tempo que aqueles apresentam melhores técnicas de fabrico o que origina uma maior qualidade e maior número de pixéis por sensor. Relativamente aos aspetos negativos este tipo de sensores consome muita energia, não suporta velocidades tão elevadas como o CMOS e são mais caros. Este tipo de tecnologia não permite a integração de circuitos integrados adicionais no mesmo sensor para fazer operações adicionais como amplificação e processamento de sinal. Nos dias que correm é o tipo de sensor mais utilizado.

### 2.3.2 Sensores CMOS

A principal diferença entre os dois sensores está na forma como os valores de luz são lidos. No caso dos CMOS [15] cada linha de sensores pode ser selecionada diretamente para ser lida. Neste aspeto o funcionamento de um sensor CMOS é muito similar ao funcionamento de uma RAM. Além disso cada sensor de cada pixel possui um amplificador diretamente acoplado. A imagem seguinte ilustra o esquema dos CMOS.

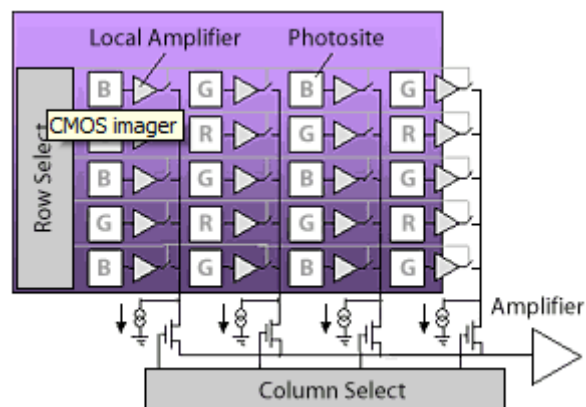


Figura 2.3 - Esquema física dos sensores CMOS. [16]

Hoje em dia, a qualidade de imagem dos sensores CMOS [17] está muito próxima à dos CCD em formatos pequenos e médios. Em formatos superiores os CCD continuam a ter uma qualidade muito superior. Devido ao facto de os sensores CMOS poderem incorporar outros tipos de circuitos no mesmo *chip* consegue-se obter sensores mais compactos, com menor consumo de potência e mais baratos. Contudo a junção de vários componentes no mesmo *chip* resulta na diminuição da sensibilidade nestes sensores, devido à área ocupada por elementos não fotossensíveis. Isto resulta necessariamente num maior tempo de exposição de consequentemente num aumento de ruído. A figura seguinte ilustra a diminuição da sensibilidade em sensores CMOS.

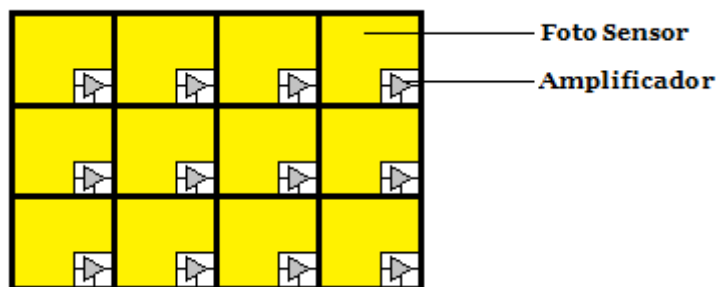


Figura 2.4 - Incorporação de circuitos no sensor CMOS.

Para quantificar a quantidade de área ocupada pelos circuitos foto sensíveis é utilizado o fator aproveitamento (*Fill Factor*) que é dado pela divisão da área foto sensível pela área total.

Existe dois tipos de sensores CMOS [15]: os ativos possuem um circuito em cada pixel que determina o nível de ruído em cada pixel e elimina-o e os sensores passivos que não possuem esta funcionalidade.

## 2.4 Processamento Distribuído em Aplicações Robóticas

Existem duas principais formas de abordagem à organização dos processadores de um sistema robótico: Processamento Centralizado e Processamento Distribuído. Os sistemas com a primeira das formas, apesar de serem de mais fácil implementação impossibilitam a existência de processamentos paralelos e a inoperância total do sistema se a unidade de processamento deixar de funcionar.

Em sistemas de visão artificial o objeto que vai ser submetido a processamento é uma imagem. As imagens são pacotes de dados que têm tamanho substancial. Em sistemas de processamento central estas imagens precisam ser transferidas até ao dispositivo que fará o seu processamento. Apesar de existirem redes comunicação que permitem a transmissão a alta velocidade, o tempo de transmissão de uma imagem não é desprezável. Podem ser utilizadas técnicas de compressão nas imagens no entanto, uma compressão significativa terá como consequência a perda de informação. Para além de existir o problema da transferência de dados existe também o problema da carga de processamento se o sistema possuir muitos sensores.

Uma parte muito importante no projeto de um sistema com processamento distribuído é a seleção do sistema de comunicações que as várias unidades de processamento vão utilizar

para comunicar entre si. Existem vários tipos de protocolos e a escolha dos mesmos depende de fatores como:

- Previsibilidade dos tempos em que ocorre a comunicação entre os vários dispositivos.
- Níveis de tolerância a *jitter* (incerteza quanto ao tempo de transferência).
- Tamanho dos pacotes a serem transferidos.
- Sincronismo e assincronismo das mensagens.
- Importância na detecção de falha de transmissão.

Os protocolos mais utilizados neste âmbito são Ethernet industrial, CAN, Profibus e Modbus. Em casos de comunicações muito simples e com dados de pequena dimensão a serem trocados podem ser utilizados protocolos como RS232 e RS485. [18]

As principais vantagens dos sistemas de processamento distribuído relativamente ao centralizado são:

- Utilizam melhor o poder de processamento.
- Apresentam melhores desempenhos.
- Têm maior grau de confiabilidade.
- Permitem compartilhar dados e recursos.

Relativamente às desvantagens:

- Maior dificuldade em criar e manter o sistema devido à sua maior complexidade.
- Controlo do acesso aos dados por parte dos diversos dispositivos.
- Falhas locais podem afetar o sistema geral.
- Garantia da segurança do sistema.

Relativamente às *smart cameras* [19] estas são capazes de efetuar a sensorização do meio envolvente, efetuam o processamento das imagens adquiridas e são capazes de lidar com protocolos de comunicações pelo que são sistemas altamente adaptados a integrar sistemas de processamento distribuído.

## 2.5 Processamento de imagem

O processamento de imagem é o que nos permite obter os parâmetros matemáticos do que se pretende extrair de uma imagem. Genericamente pode dizer-se que existem três fases neste processamento:

- Pré-processamento - Permite preparar a imagem para o processamento posterior.
- Processamento propriamente dito - Nesta fase do processo extrai-se as características da imagem.
- Pós-Processamento - Permite validar as características anteriormente detetadas e fazer tratamento de dados obtidos pelo processo anterior e fornecê-los ao sistema de controlo superior.

Nesta secção deste capítulo pretende-se fazer uma apresentação teórica de algumas das técnicas que poderão vir a ser úteis no âmbito desta dissertação. A apresentação dessas técnicas encontra-se dividida pelas três fases do processamento anteriormente apresentada.

## 2.5.1 Pré-processamento

O pré-processamento tem como principal objetivo fazer a preparação de imagem para processamento posterior. Estas tarefas podem ser de realce de orlas, filtragem, eliminação de ruído, alteração do tamanho da imagem, etc.

### 2.5.1.1 Filtragem

O processo de filtragem utiliza filtros lineares que mais não são do que matrizes de convolução.

Os filtros lineares mantêm o nível médio de todos os pixéis de imagem, os filtros não lineares aplicam transformações à imagem sem o compromisso de manterem o nível médio dos valores de todos os pixéis.

Um filtro numérico influencia a variação da frequência espacial de uma imagem. Enquanto na frequência temporal se utiliza como unidade o Hertz, em processamento de imagem a frequência espacial é dada por 1/metro ou 1/pixel. A frequência espacial descreve a velocidade de modificação da luminosidade ou cor em uma direção de uma imagem.

A filtragem é um processo aplicado pixel a pixel e onde o valor de cada um deles depende dos valores da vizinhança. Em geral os pontos mais próximos de cada pixel contribuem mais para o valor do pixel do que os pontos mais afastados.

Para ser realizada a operação de filtragem primeiramente é necessário escolher uma matriz de dimensão  $m \times m$ , normalmente 3x3 ou 4x4. Os valores da matriz dependem do tipo de filtro que se quer usar.

O ruído corresponde a modificações abruptas dos níveis de cor ou cromaticidade. Se for pretendido eliminar este ruído deve-se utilizar filtros passa-baixo que irão eliminar as altas frequências (que correspondem ao ruído).

Um filtro clássico [20] é a utilização de uma matriz com os elementos todos iguais a 1. Cada pixel resultante da convolução será igual à média do próprio com os 8 pixéis envolventes na imagem original.

Apresentam-se agora uma imagem que foi submetida ao filtro com todos os elementos da matriz 3x3 iguais a 1.



Figura 2.5 - Resultado da aplicação do filtro.

Pode-se verificar que os pontos brancos aleatórios adicionados aleatoriamente para simulação de ruído na imagem à direita estão mais suavizados.

### 2.5.1.2 Detetores de contornos

Os detetores de contornos detetam variações bruscas tanto de luminosidade como de cor. Os detetores de contornos não são mais que filtros específicos que fazem a convolução discreta. De seguida apresentam-se alguns tipos de detetores de orlas que poderão ter interesse no desenrolar da dissertação.

#### **Operador Prewitt**

Este operador [21] baseia-se no cálculo da aproximação do valor do gradiente. Existe uma filtragem responsável por calcular a derivada parcial em ordem ao eixo dos x quer irá salientar as arestas horizontais e outra filtragem responsável por calcular a derivada parcial em ordem ao eixo dos yy que irá salientar as arestas verticais.

$$\nabla_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \nabla_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Valores dos elementos das matrizes do Operador Prewitt.

Com os valores das derivadas parciais é possível calcular a magnitude e os ângulos do gradiente pelas seguintes fórmulas:

$$|\nabla f| = \left[ \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 \right]^{\frac{1}{2}} \quad (2.4) \quad \theta = \text{tg}^{-1} \left[ \frac{\nabla_y}{\nabla_x} \right] \quad (2.5)$$

#### **Operador Sobel**

O funcionamento deste operador [21] é muito semelhante ao Prewitt apenas se alterando os valores dos elementos das matrizes.

$$\nabla_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \nabla_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

#### **Operador Canny**

Este operador [21] foi desenvolvido por John Canny em 1986 e teve os seguintes objetivos:

- Boa deteção - O operador tem de salientar todas as arestas existentes na imagem.

- Boa localização - As arestas salientadas tem de estar o mais próximo possível das arestas da imagem original.
- Resposta mínima - Cada aresta da imagem deve ser salientada apenas uma vez, ou seja, deve ter apenas um pixel de largura. O ruído existente na imagem também não deve de criar falsas arestas não existentes na imagem original.

Os valores dos elementos da matriz são os seguintes:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

### **2.5.1.3 Binarização**

Para analisar uma imagem as variações nos valores de intensidade luminosa devem ser interpretadas. A gama de valores que permitem representar a luminosidade de uma imagem é de 256 valores diferentes no caso da *Surveyor Blackfin Camera*. Estes valores diferentes implicam um maior esforço computacional pelo que é aconselhável usar técnicas de binarização.

Para se realizar esta operação escolhe-se um valor limiar para a imagem e compara-se esse valor com os valores dos pixéis da imagem. Se o valor do pixel da imagem for superior ao limiar esse pixel toma o valor da luminosidade máxima, se for inferior toma o valor da luminosidade mínima.

A binarização [22] é uma das técnicas mais usadas para aplicações em tempo real porque reduzem em muito os tempos de processamento. Quando a imagem tem mais de dois objetos com valores de luminosidade diferentes utiliza-se a técnica de binarização multinível [23]. Nesta são escolhidos dois ou mais limiares e 3 ou mais valores que os pixéis devem tomar. Apesar de a binarização ser um processo simples de implementar a escolha dos limiares é complexa.

## **2.5.2 Processamento**

A fase de processamento de imagem propriamente dita corresponde a extrair características da imagem. Estas podem ser formas geométricas que podem ser parametrizadas matematicamente, caracteres, deteção de cores, etc. No nosso caso específico e como se pretende a deteção de linhas apresentam-se aqui dois algoritmos que podem ser úteis para o desenvolver do projeto.

### **2.5.2.1 Transformada e Hough**

A transformada de Hough [24] é uma técnica matemática utilizada para deteção de formas geométricas. A versão original construída por Paul Hough em 1962 era apenas capaz de fazer a deteção de retas. Com trabalhos de outros autores os princípios da transformada

de Hough original foram sendo aplicados em outras vertentes e tornaram possível a detecção de outras figuras geométricas como círculos e elipses.

Nesta subsecção descreve-se o algoritmo da HT para a detecção de retas.

O cerne da HT está em definir um mapeamento entre o espaço de imagem e o espaço de parâmetros. Normalmente é utilizada depois de ser aplicado um detetor de arestas à imagem (pré-processamento).

A transformada de Hough consiste em mapear cada pixel de uma borda no espaço de parâmetros, que é organizado em forma de acumulador.

Para definir uma reta  $y=mx + b$  é preciso dois parâmetros  $m$  e  $b$ . Claro que se houver retas paralelas ao eixo das ordenadas  $m$  terá um valor infinito. Para simplificar a explicação do processo HT considera-se que não irão existir retas com declive infinito. Posteriormente, será apresentada uma técnica que permite contornar este problema.

Um exemplo de um acumulador para a detecção de retas é o seguinte:

**Tabela 2.3 - Exemplo de acumulador para a detecção de retas.**

| <b>M</b> | <b>b</b> | <b>Contador</b> |
|----------|----------|-----------------|
| 1        | -3       | 2               |
| 1        | -2       | 3               |
| 1        | -1       | 1               |
| 1        | 0        | 3               |
| 1        | 1        | 2               |
| 1        | 2        | 3               |
| 1        | 3        | 6               |

Para calcular os parâmetros  $m$  e  $b$  são necessários dois pontos diferentes da imagem,  $m = \frac{y_2 - y_1}{x_2 - x_1}$  e  $b = y_1 - m * x_1$ .

O algoritmo da transformada de Hough [25] percorre todos os pixéis de informação existentes na imagem (pixéis diferentes do fundo resultantes do pré-processamento) e calcula as equações da reta a partir da seleção de dois pontos. De seguida é incrementado o valor do contador na posição correspondente aos valores de  $b$  e  $m$  calculados anteriormente. Repara-se que se os dois pontos forem selecionados de retas diferentes ou de pontos de ruído, é muito improvável que estes formem os mesmos valores de  $m$  e  $b$  várias vezes. Assim sendo os contadores destas posições terão valores baixos porque raramente serão detetados repetidamente os mesmos valores de  $m$  e  $b$  quando os dois pontos selecionados pertencem a retas diferentes e/ou a pontos de ruído.

No fim do algoritmo percorre-se todas as posições do acumulador e verifica-se quais os valores de  $m$  e  $b$  cujo valor do contador é superior a um limiar previamente definido. Esses valores de  $m$  e  $b$  selecionados anteriormente têm grande probabilidade de definirem retas da imagem. Através do pós-processamento é possível verificar onde começam e acabam as retas se isso for importante, verificar se as retas detetadas existem mesmo, procurar intersecções, etc.

Para contornar o problema das retas verticais já anteriormente referido foi inventada uma nova forma de parametrizar as retas, utilizando coordenadas polares [26].

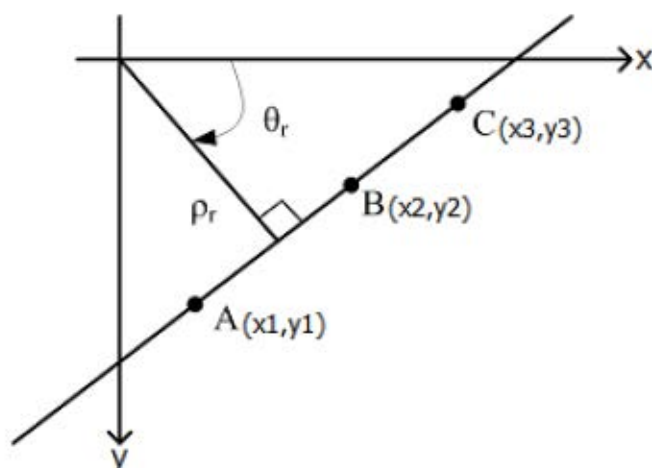


Figura 2.6 - Representação das retas em coordenadas polares. [4]

Utilizado o comprimento  $\rho_r$  e o ângulo  $\theta_r$  da reta normal à reta a ser detetada consegue-se parametrizar completamente as retas existentes na imagem. Todos os pontos da reta detetada com os parâmetros  $\rho_r$  e  $\theta_r$  satisfazem a seguinte equação:

$$\rho_r = x1 * \cos(\theta_r) + y1 * \sin(\theta_r) = x2 * \cos(\theta_r) + y2 * \sin(\theta_r) \quad (2.6)$$

A transformada de Hough é altamente robusta e quase imune ao ruído, no entanto apresenta um custo computacional muito elevado que nem todos os processadores podem suportar. Ao longo dos tempos foram criados vários tipos de variantes da transformada com o objetivo de minimizar o tempo de processamento requerido para completar o algoritmo. Apresentam-se de seguida algumas das versões mais utilizadas da transformada de Hough.

#### Transformada de Hough Probabilística

Esta variante [27] foi inventada em 1990. Consiste em calcular os parâmetros  $\rho_r$  e  $\theta_r$  apenas de uma proporção dos pixéis da imagem, existindo uma escolha de pixéis baseada numa função densidade de probabilidade que varia consoante a imagem que se pretende analisar. Desta forma apenas os pixéis que têm uma probabilidade muito alta de pertencerem a uma reta são utilizados para o cálculo dos parâmetros.

#### Transformada de Hough Aleatória

Este método [28] está baseado no facto de que um ponto no novo espaço de parametrização pode ser determinado a partir de um pequeno subconjunto de pontos de uma aresta da imagem original. O tamanho deste subconjunto depende exclusivamente da complexidade das curvas a serem detetadas. Após cada incrementação do contador é verificado se o contador ultrapassou um determinado limiar. Se este limiar for ultrapassado é definida uma nova reta e esta reta é eliminada da imagem, resultando numa nova imagem com menos pontos de informação e menos retas para serem detetadas, o que acelera o processo.

### Transformada de Hough Hierárquica

Esta transformada [29] foi criada em 1989 e baseia-se numa estrutura em pirâmide onde cada camada da mesma divide a imagem em subimagens. A pirâmide *standard* tem  $P+1$  níveis onde  $P$  é o nível superior da imagem que representa a imagem total (todas as formas nelas existentes). No nível mais baixo são detetados pequenos segmentos de reta em uma subimagem o que origina uma diferenciação entre linhas reais e pontos casualmente alinhados como ruído. A eficiência do método prende-se com o facto de ser usado uma matriz acumuladora pequena para representar as linhas em cada sub-imagem.

### Transformada de Hough Combinatorial

Esta variante [29] evita o processo de votação da transformada de Hough original. Para cada par de pontos são calculadas todas as possíveis combinações de segmentos de linha entre os dois pontos. Para o caso da deteção de retas, são precisos dois parâmetros para definir cada uma delas. O limiar da deteção é dado por:

$$C_m^2 = \frac{m!}{2!(m-2)!} \quad (2.7)$$

Em que  $m$  é o número mínimo de pontos sobre uma reta para que a mesma possa ser considerada como tal.

#### 2.5.2.2 EDlines

O algoritmo EDlines é uma técnica inventada muito recentemente (ano de 2011) para fazer a deteção de retas e segundo testes do autor é o algoritmo de deteção de linhas mais rápido.

Os métodos tradicionais geralmente utilizam o filtro Canny para salientar as arestas e depois utilizam algoritmos como a transformada de Hough para parametrizar as retas. Isto dá origem a um processo demorado e podem ser detetadas retas que não são contínuas (na verdade são duas retas) e retas falsas. Este método foi desenvolvido de maneira a eliminar as desvantagens anteriores.

Este algoritmo pode ser dividido em 2 fases diferentes:

1. Através de uma imagem em tons de cinzento é aplicado o algoritmo *Edge Drawing* que produz uma lista de segmentos de reta.
2. Depois calcula-se os parâmetros matemáticos de todos os segmentos de reta presentes na lista elaborada anteriormente. Para isso utiliza-se o método dos mínimos quadrados.

Começa-se por fazer uma explicação acerca do método *Edge Drawing* [30] que é o mais importante deste algoritmo.

## ❖ Primeira etapa

### Edge Drawing

A maioria dos detetores de arestas tradicionais limitam-se a eliminar os pixéis que não são pertencentes a uma aresta. Este algoritmo produz vetores com as coordenadas das arestas detetadas.

Os principais objetivos do ED são os seguintes:

1. Poder ser processado em tempo real.
2. Produzir *edges* contínuos, bem localizados e com apenas 1 pixel de largura.

Pode-se dizer que este algoritmo é realizado em 4 passos:

1. Eliminação do ruído através da convolução da imagem com um filtro gaussiano.
2. Cálculo do direção e magnitude do gradiente.
3. Extração de pontos de interesse, definidos como *anchors*.
4. Ligação dos *anchors* através de um método de *smart routing*.

### ➤ Primeiro Passo

É feita a convolução da imagem com um filtro Gaussiano. Utiliza-se uma matriz com todos os seus elementos iguais a 1. O tamanho da matriz a utilizar depende do tipo de aplicação. De notar que este algoritmo trabalha apenas com imagens de tons de cinzento. À saída deste passo tem-se uma imagem com o ruído eliminado.

### ➤ Segundo Passo

É calculada a magnitude e direção do gradiente, para isso pode ser utilizado tanto o operador Sobel como o operador Prewitt ou outros similares. A figura seguinte ilustra o princípio do funcionamento deste passo.

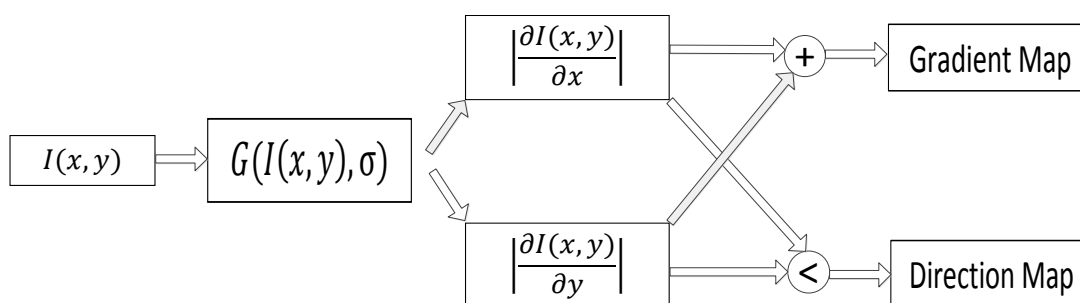


Figura 2.5 - Ilustração do funcionamento do passo nº 2. [31]

Como *outputs* deste passo temos duas matrizes: o mapa de gradiente que não é mais do que a magnitude do gradiente e o mapa de direção que diz que o pixel pertence a uma aresta vertical ou horizontal.

A magnitude do gradiente (**G**) num determinado ponto é igual á soma das derivadas parciais em x e em y.

$$G = \frac{|\partial I|}{|\partial x|} + \frac{|\partial I|}{|\partial y|} \quad (2.8)$$

A direção do gradiente (**D**) é dada pela comparação dos valores das derivadas parciais.

Se  $\frac{|\partial I|}{|\partial x|} \geq \frac{|\partial I|}{|\partial y|}$ , então considera-se que por este pixel passa uma aresta vertical e **D** toma o valor de 0.

Se  $\frac{|\partial I|}{|\partial x|} < \frac{|\partial I|}{|\partial y|}$ , então considera-se que por este pixel passa uma aresta horizontal e **D** toma o valor de 1.

A direção do gradiente tomará apenas dois valores possíveis 0 e 1 que correspondem a 0° e 90° respetivamente. De acordo com testes do autor deste algoritmo usando outros valores de ângulo diferentes aumenta o tempo de processamento total, pelo que o uso de dois valores diferentes é suficiente.

Para além do cálculo da magnitude e direção do gradiente é ainda efetuada uma binarização ao nível da magnitude. Todos os valores de magnitude abaixo do limiar = 36 são considerados zero. O valor deste limiar pode variar consoante o tipo de aplicações, género de imagens, etc.

### ➤ Terceiro passo

Até agora este algoritmo é muito semelhante a todos os outros já existentes de deteção de arestas contudo a partir deste passo diferencia-se. Em vez de testar cada pixel da imagem calcula uma lista de pontos que têm alta probabilidade de pertencerem a uma aresta (*anchors*) e a sua respetiva direção (horizontal ou vertical). Este processo tem a vantagem de não ser necessário percorrer todas as linhas e colunas à procura de *anchors* uma vez que uma linha contém mais do que um *anchor*. No entanto se todos os detalhes de uma imagem forem importantes é necessário percorrer todas as linhas e todas as colunas de pixéis.

Se o pixel tiver no mapa de direção uma direção Horizontal testa-se a seguinte condição1:

**Condição 1:**  $G[x, y] - G[x, y-1] \geq \text{ANCHOR\_THRESH} \ \&\& \ G[x, y] - G[x, y+1] \geq \text{ANCHOR\_THRESH}$

Se por sua vez tiver uma direção Vertical testa-se a condição 2:

**Condição 2:**  $G[x, y] - G[x-1, y] \geq \text{ANCHOR\_THRESH} \ \&\& \ G[x, y] - G[x+1, y] \geq \text{ANCHOR\_THRESH}$

Apresenta-se agora um esquema ilustrativo da deteção de *anchors* para cada *anchor*.

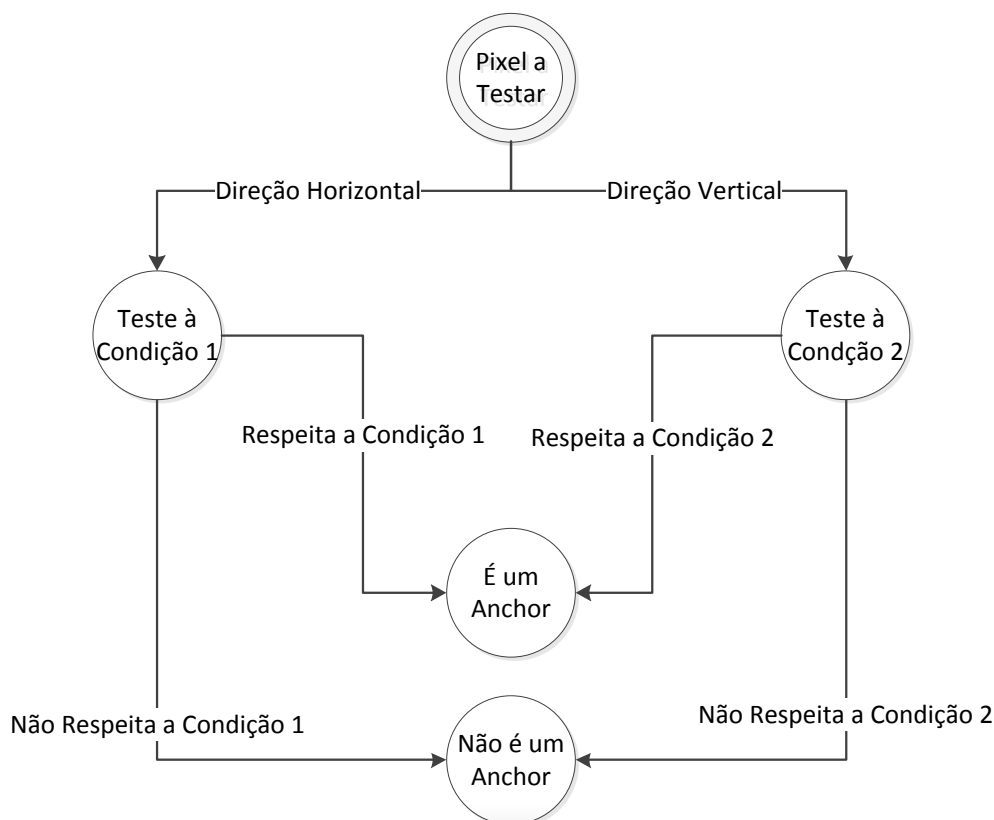


**Figura 2.7 - Esquema ilustrativo da detecção de *anchors*.**

Posto isto, é criado um vetor com todos os *anchors* anteriormente detetados.

O limiar `ANCHOR_THRESH` é por norma igual a 8. Mais uma vez dependendo do tipo de aplicações e do género de imagens este valor pode ser alterado. Por sua vez, se houver necessidade de encontrar mais *anchors* o valor deve ser menor, se houver necessidade de encontrar menos e consecutivamente diminuir o tempo de processamento o valor deve ser maior.

#### ➤ Quarto passo

Depois de estar definido o vetor de *anchors* a etapa seguinte é a partir de cada *anchor* fazer a sua ligação aos outros *anchors*. É neste momento que se desenhavam as várias retas e se guarda os pixels pertencentes à reta num vetor. Para o fazer são verificados os valores dos 3 pixels adjacentes e escolhido o pixel com maior valor em termos de luminância.

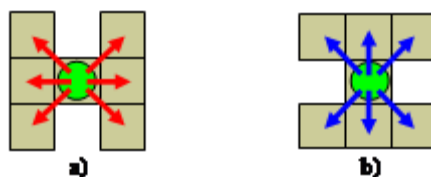


**Figura 2.8 - a) Pesquisa por arestas horizontais; b) Pesquisa por arestas verticais. [31]**

De seguida é feita uma explicação deste processo através de um código. Apresenta-se apenas o caso de um *anchor* horizontal e a desenhar a reta para a esquerda. De notar que

depois de desenhar a reta para a esquerda volta-se ao *anchor* e desenha-se o resto da reta existente para a direita. O procedimento para desenhar a reta noutras direções é completamente análogo ao apresentado.

Escolhendo um *anchor* da lista criada no terceiro passo o procedimento seguinte é este:

```
while (G[x, y] > 0 && D[x, y] == HORIZONTAL && E[x, y] != EDGE){
    E[x, y] = EDGE;          // marcar o pixel como um pixel pertencente a uma reta e
                             // guarda-lo num vetor

    // verifica os valores dos três pixeis à esquerda
    if ([x-1, y-1] > G[x-1, y] && G[x-1, y-1] > G[x-1, y+1]){
        x = x-1; y = y-1;    // é escolhido o pixel superior esquerdo
    }
    else if (G[x-1, y+1] > G[x-1, y] && G[x-1, y+1] > G[x-1, y-1]){
        x = x-1; y = y+1;    // é escolhido o pixel inferior esquerdo
    }
    else {
        x = x-1;              // é escolhido o pixel à esquerda
    }
}
```

Como é perceptível pelo código acima transcrito o algoritmo de desenho da reta pára quando é encontrado um pixel de fundo (com valor igual a 0 de luminosidade) ou quando é encontrado um pixel que faz parte de uma direção diferente da reta que esta a ser desenhada no momento ou ainda quando o pixel atual já foi caracterizado como pertencente a uma reta. De notar que ao longo deste processo as coordenadas (valores de *x* e *y*) pertencentes à reta são guardadas num vetor para processamento futuro. Para cada reta encontrada é criado um novo vetor. A figura seguinte exemplifica o processo de desenho de uma reta.

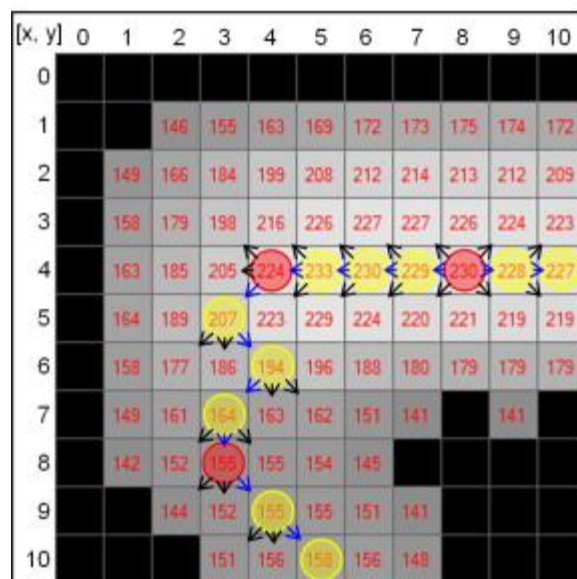


Figura 2.9 - Exemplificação do funcionamento do *smart routing*. [31]

Como *output* deste processo tem-se os vetores com as coordenadas dos pixéis que correspondem às retas detetadas. E assim se conclui a primeira etapa do EDlines. Agora os vetores anteriormente referenciados irão sofrer um processamento na próxima etapa.

#### ❖ Segunda etapa

Com os vetores anteriormente criados e a partir das coordenadas existentes nos mesmos é efetuado o cálculo de uma parametrização para as retas através do método dos mínimos quadrados.

E assim se termina o algoritmo do EDLINES.

### Pós-processamento

O pós-processamento é a última parte do processamento de imagens mas não menos importante. É uma fase em que nos permite corrigir erros e verificar a validade das características extraídas. No caso da deteção de retas pode-se conferir pontos de intersecção, determinar o início e o fim da reta, determinar a orientação, etc..

É também nesta fase que se faz o tratamento dos dados para os fornecer depois ao sistema de controlo superior.



## Capítulo 3

### Sistema em estudo

Este capítulo resulta da materialização do primeiro e segundo objetivo desta dissertação: estudar o sistema e criar uma documentação acerca do mesmo que permita uma rápida curva de aprendizagem a quem posteriormente decida trabalhar com a *Surveyor Blackfin Camera*.

A *Surveyor Blackfin Camera* é uma *smart camera* fabricada pela empresa *Surveyor Corporation* sediada nos Estados Unidos da América, mais precisamente na Califórnia. Foi fundada em 1988 por Howard Gordon [32] e tem vindo a dedicar-se essencialmente ao fabrico de sistemas robóticos *open source*.

Esta câmara é normalmente vendida como peça integrante de um pequeno robô produzido pela mesma empresa direccionado para fins educacionais e de investigação. Este pequeno robô como ilustra a figura seguinte é constituído pela *smart camera* em estudo, por um módulo de comunicações *wireless*, quatro motores para a deslocação do sistema e dois *pointing lasers*. Podem ainda ser acoplados muitos outros módulos a este sistema, os mais comuns são os módulos de: gps, comunicação gsm, cartões SD.



Figura 3.1 - Robô onde se insere a câmara em estudo. [39]

No entanto existe a opção de comprar apenas a *Surveyor Blackfin Camera*. O *firmware* disponibilizado pelo fabricante é criado para o controle e comunicação deste robô anteriormente apresentado. Não existe nenhum *firmware* criado especificamente para a

câmara, mesmo assim, este tem todas as funcionalidades necessárias para trabalhar com aquela.

Neste capítulo começa-se por fazer uma descrição do *hardware* do sistema, depois uma descrição sobre a arquitetura do *software* (*firmware* disponibilizado pelo fabricante), uma enumeração das principais funções relacionadas com a aquisição e processamento de imagem e das formas de interação com o sistema.

Para além da documentação presente neste capítulo foram ainda produzidos tutoriais para a configuração e programação do sistema e encontram-se na parte final deste relatório na secção de anexos.

## 3.1 Descrição do Hardware

### 3.1.1 Sensor de Imagem OV7725

É uma câmara VGA [33] com alta sensibilidade para operar em condições de pouca luz, possui uma interface SCCB, capaz de operar a 60 *frames* por segundo. É fabricado segundo a tecnologia CMOS. Tem as seguintes especificações:

- 640 x 480 pixels de resolução;
- Pode ser alimentado por fontes CC entre 3,0 V e 3,3 V;
- Potência consumida: 120 mW (funcionamento máximo) 60  $\mu$ W (quando em standby);
- Tamanho da lente: 0,635 cm;
- Funciona de -20°C a 70°C.

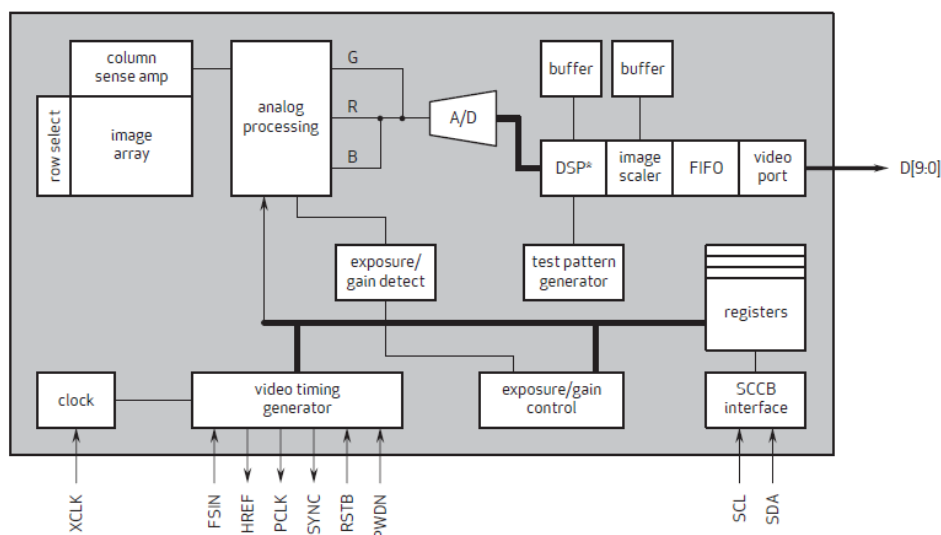


Figura 3.2 - Diagrama de blocos funcionais do sensor de imagem. [40]

O sensor de imagem propriamente dito está conectado a um bloco de processamento analógico que faz algumas correções ao nível de ganho, brilho, contraste, etc. Seguidamente o conversor analógico digital converte a imagem para um formato digital. A DSP efetua

correções à imagem tais como: correção de brancos, eliminação de ruído, etc. A imagem depois passar por um bloco que irá alterar o tamanho da mesma, segue para um *buffer FIFO* (*first in first out*) e por fim para a *video port* onde será transferida para o processador através das portas digitais.

O bloco SCCB permite obter a imagem sem processamento (*raw RGB*) e ainda ajustar certos parâmetros como o ganho, brilho, *frame rate*, e formato de saída da imagem através da alteração dos registos. É composto por dois sinais SCL (sinal *clock*) e SDA (sinal de dados).

O bloco de *video timing generator* cria o sinal que irá sincronizar a sensor de imagem com o processador BlackFin para a recepção de vídeo.

O bloco de *Clock* recebe o sinal do cristal e fornece o sinal de relógio ao sistema.

### 3.1.2 Microprocessador Analog Devices Blackfin BF537

É um processador com uma arquitetura resultante da fusão de duas linhas arquitetónicas: RISC e a SIMD. RISC (*Reduced Instruction Set Computer*, Computador com um Conjunto Reduzido de Instruções) é uma linha de arquitetura de processadores composta por um conjunto pequeno de instruções de baixa simplicidade que têm aproximadamente o mesmo tempo de processamento. SIMD (*Single Instruction Multiple Data*) é um método de operação computacional em que uma instrução é simultaneamente aplicada a diferentes conjuntos de dados, no caso deste sistema este método é muito utilizado nas operações multimédia.

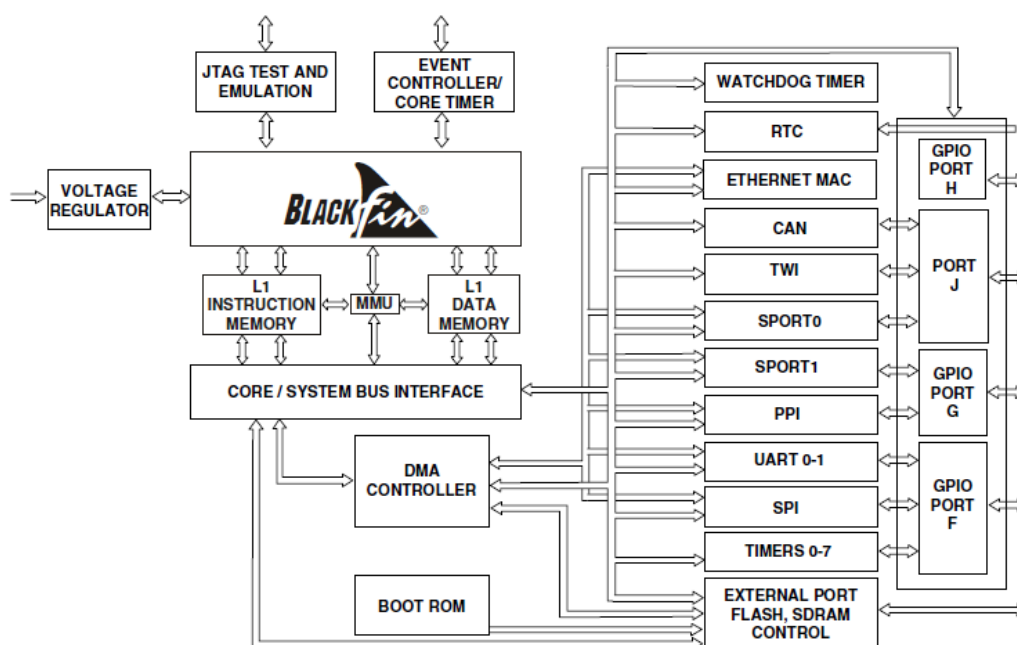


Figura 3.3 - Diagrama de blocos do processador. [41]

O núcleo do processador está conectado a uma série de interfaces muito utilizados na indústria, atualmente, tornando-o bastante versátil. A comunicação com outros sistemas pode ser feita através de Ethernet, CAN, TWI, das 2 portas série disponíveis (UART), porta

SPI, 9 timers (8 dos quais podendo ser modulados por PWM), e uma PPI (*parallel peripheral interface*). Faz-se agora uma descrição dos diferentes blocos funcionais.

- **Portas Série (SPORTs, Synchronous Serial Ports)**

Existem duas portas séries síncronas (SPORT0 e SPORT1) para comunicações síncronas com outros processadores ou periféricos. De seguida apresenta-se as suas principais características:

- Suporta comunicação *I<sup>2</sup>S* (*Integrated Interchip Sound*). É um protocolo de comunicação série normalmente utilizado para conectar dispositivos de áudio digitais. Caracteriza-se por ter um *jitter* muito baixo.
- Pode trabalhar com um sinal de relógio externo mas tem também a capacidade de gerar o seu próprio sinal de *clock*.
- Suporta transmissão de *words* de 3 a 32 bits, podendo seleccionar se o bit mais ou significativo ou o menos significativo é transferido primeiro.
- Cada porto de transmissão ou receção pode trabalhar com ou sem sinal de sincronização. Este sinal de pode ser gerado internamente ou externamente.
- Compatível com os *standards* H.100, H.110, MVIP-90 e HVIP.

- **Porto SPI (Serial Peripheral Interface)**

Permite comunicar em modo síncrono com periféricos que suportem este *standard* em modo *full-duplex*. Utiliza 3 *pins*: MOSI (*Master Output, Slave Input*), MISO (*Master Input, Slave Output*) e o SCK (*Serial Clock*). Pode operar em modo *master* e também como *slave*.

- **Porto UART (Universal Asynchronous Receiver and Transmitter)**

Existem dois portos UART que suportam comunicação *full-duplex*. Em cada trama permite transferir entre 5 a 8 *bits* de dados, até 2 stop bits, nenhum ou um bit de paridade (par ou ímpar). Estes portos têm uma prioridade baixa no acesso à memória visto que costumam ter uma taxa de utilização da memória mais baixa quando comparados com comunicações síncronas.

- **CAN (Controller Area Network)**

Através de um controlador CAN é possível implementar o protocolo CAN 2.0B. Este protocolo muito usado na indústria automóvel permite efetuar comunicações assíncronas. As características elétricas de cada rede CAN variam muito, e como tal, a interface CAN é tipicamente dividida em 2 partes: o controlador e o transceptor. O controlador está incorporado com o processador, o transceptor deverá ser adquirido consoante as características da rede a conectar.

- **TWI (Two Wire Interface)**

Este módulo é compatível com  $I^2C$  e permite fazer a comunicação SCCB com a câmara podendo-se desta forma alterar os registos da mesma.

- **ETHERNET MAC**

O processador tem a capacidade de conectar diretamente a uma rede Ethernet através de um mecanismo de controlo de acesso ao meio embutido. Suporta operações a 10 e a 100 Mbps e é compatível com a norma IEEE 802.3-2002.

- **Portos**

Existem 4 grupos de portos: Port F, Port G, Port H, Port J. Existem 8 *pins* (Port F0-7) que têm uma capacidade de operar em correntes mais altas que os restantes *pins*.

- **General-Purpose I/O (GPIO)**

Existem 48 *pins* de input/output associados aos três diferentes módulos de portos (Port G, Port H e Port F). Através dos registos é possível configurar os diferentes *pins* como *input* ou *output*, alterar o valor da tensão entre outras operações básicas tais como *toggle bit*. Os “*interrupt mask registers*” que são registos que permitem configurar os vários *pins* como *interrupts* para o processador. Os “*interrupt sensivity registers*” permitem definir o funcionamento dos *interrupts*, se são sensíveis ao nível baixo ou alto, ou ao *rising* ou *falling edge*.

- **Parallel Peripheral Interface (PPI)**

Este tipo de interface paralela pode ser conectado diretamente a conversores A/D ou D/A, geralmente *encoders* e *decoders* de vídeo. É composto por 1 *input clock pin*, 3 *pins* de sincronização e 16 *pins* de data.

- **Controlador DMA**

Existem vários controladores DMA (*Direct Memory Access*) que permitem transferências de dados com interferência mínima do núcleo do processador, evitando assim a sobrecarga do mesmo. O DMA controla tanto as transferências de dados entre os componentes do processador como também as transferências de dados entre o núcleo do processador e os periféricos conectados.

- **RTC (Real-Time Clock)**

Trata-se de um relógio digital que permite saber o tempo atual, produzir um alarme, criar interrupções, etc. A sua alimentação é independente da alimentação do núcleo do processador o que permite que este bloco continue a funcionar mesmo quando o núcleo do processador está desligado. Desta forma é possível que o RTC produza interrupções que alterem o estado de funcionamento do núcleo do processador.

- **Watchdog timer**

É composto por um *timer* de 32 bits que pode ser usado para evitar que o sistema entre em estados desconhecidos ou de bloqueio. Se uma operação estiver a demorar demasiado tempo a ser realizada e o contador ao ser decrementado chegar ao seu valor nulo é possível forçar o programa a ir para um estado conhecido através de um *hardware reset*, interrupt, etc.

- **Núcleo do processador**

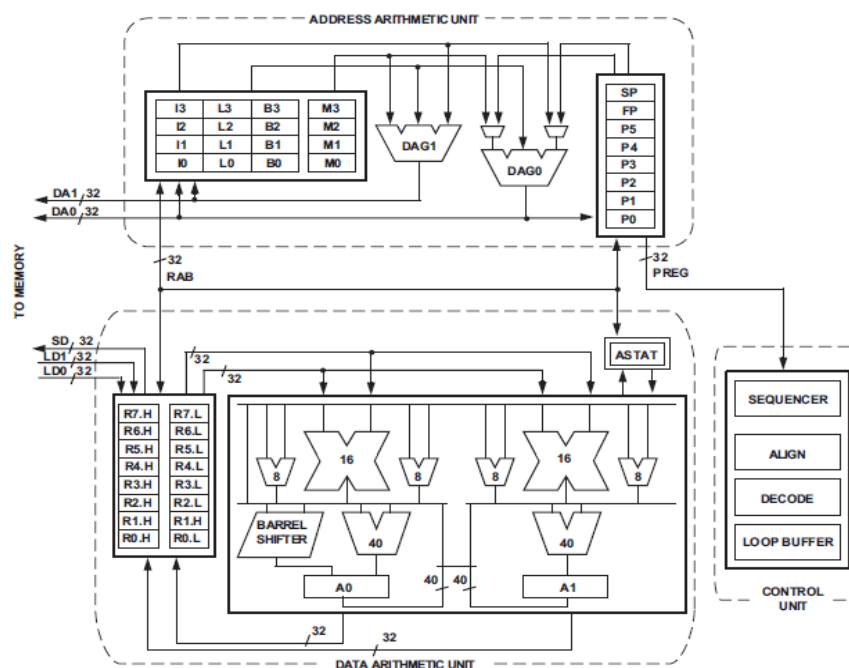


Figura 3.4 - Núcleo do processador. [41]

O núcleo do processador [34] é composto por dois multiplicadores de 16 *bits*, dois acumuladores de 40 *bits*, duas ALUs de 40 *bits* quatro ALUs de vídeo e um *shifter* de 40 bits. O banco de registo de cálculos é composto por registos 8 de 32 *bits*. Além das operações características, as ALUs suportam instruções especiais para um processamento de sinal mais rápido, nomeado de imagem.

### 3.1.3 RAM 32MB SDRAM

O sistema tem uma memória RAM externa de 256 Mb (32 MB) K4S561632J da SAMSUNG [35]. Tem as seguintes especificações:

- Fonte de alimentação JEDEC 3,3V;
- Compatível com interface LVTTTL;
- Está dividida em 4 bancos de memória;
- Todos os *inputs* são lidos no *rising edge* do sinal de *clock* do sistema;
- *Refresh* de memória automático;
- Compatível com a diretiva RoHS.

### 3.1.4 SPI Flash

A memória Flash 4 MB [36] comunica com processador através da interface SPI. Tem as seguintes especificações:

- 32 Mbit (4 MB) de memória;
- Programação de 256 Bytes em 1,4 ms;
- Alimentada entre 2,7 V e 3,6 V;
- Compatível com interface SPI;
- Possibilidade de trabalhar até 50MHz;
- Mais de 100 000 ciclos de reprogramação;

### 3.1.5 Sistema de geração de reset

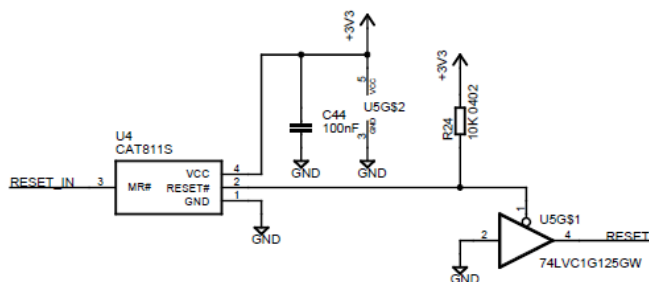


Figura 3.5 - Esquemático do sistema de geração de *reset*. [3]

O sistema de geração de *reset* além de efetuar um *reset* ao sistema a pedido do utilizador também faz *reset* ao sistema se a tensão de alimentação estiver abaixo de um certo nível. Isto irá proteger o sistema de correntes muito elevadas que possam vir a danificar os componentes do sistema, porque se a tensão for mais baixa a corrente irá aumentar para manter a mesma potência entregue. Esta verificação de tensão é feita pelo integrado CAT811S.

### 3.1.6 Slot para interface JTAG

Este *slot* para além de permitir a programação do sistema permite emular o sistema. Normalmente a programação é feita através do *bootloader*, mas se o *firmware* for danificado é necessário utilizar este *slot* para programar o sistema. Para isso é necessário ter um adaptador JTAG.

Através de um *software* disponibilizado pelo fabricante é possível fazer a emulação do sistema, podendo-se monitorizar e modificar a memória e os registos do processador.

### 3.1.7 Regulador de tensão de alimentação

A *smart camera* funciona a 3,3 V. Para evitar problemas com a tensão de alimentação existe um regulador de tensão à entrada do sistema que aceita tensões de entrada entre 4,75 V e 18 V e tem uma saída de tensão constante a 3,3V. O sistema de regulação de alimentação é feito através do circuito integrado RECOM R-783.3-1.0. Destaca-se pela sua alta eficiência, tamanho compacto e baixo *ripple*.

Com todos os elementos de *hardware* descritos conclui-se esta secção.

## 3.2 Descrição do Software

Nesta secção faz-se uma apresentação geral do *firmware*, descreve-se as suas principais funções e as zonas mais importantes da memória. Depois serão apresentadas os diversos algoritmos de processamento de imagem desenvolvidos pelo fabricante e vêm já implementados no *firmware* disponibilizado pelo mesmo. Todo o conteúdo aqui apresentado refere-se à versão disponibilizada pelo fabricante sem qualquer modificação. As alterações feitas ao *firmware* no âmbito desta dissertação serão descritas no Capítulo 5.

### 3.2.1 Firmware

O *firmware* encontra-se maioritariamente escrito em linguagem C, tendo apenas algumas funções matemáticas escritas em assembly por uma questão de eficiência no processamento do código e encontra-se disponível no endereço *online* do fabricante.

Como já referido anteriormente o *firmware* disponibilizado contém muitos ficheiros com funções que foram construídas para o acionamento de motores, localização gps, etc que no âmbito de interesse não têm aplicabilidade. Isto porque o sistema em estudo é constituído apenas pela câmara e pelo sistema microprocessado acoplado.

Os ficheiros que contêm funções para a aquisição, tratamento e processamento de imagem são os seguintes: *colors.c*, *jpeg.c*, *main.c*, *srv.c*. O ficheiro *colors.c* contém várias funções de processamento de imagem que mais à frente serão explicadas. O *jpeg.c* congrega

um conjunto de funções que permitem fazer a codificação das imagens para estas serem posteriormente enviadas pela porta série ou pelo módulo de comunicações *wireless* para o computador. Os ficheiros *main.c* e *srv.c* contêm principalmente a chamadas das funções agrupadas para produzirem os diferentes processamentos de imagem já implementados pelo fabricante. Para além disso no ficheiro *srv.c* estão as funções de aquisição de imagem.

Para a configuração da câmara, da memória e das transferências os ficheiros mais importantes são os seguintes: *camera.c*, *config.h*, *uart.c* e *XMODEM.c*. As funções de inicialização, reconfiguração dos diversos parâmetros do sensor de imagem OV7725 encontram-se disponíveis no ficheiro *camera.c*. O ficheiro *config.h* contém os valores dos principais parâmetros da câmara tais como: *MASTER\_CLOCK*, *CCLK\_DIVIDER* e os valores de *baudrate* das duas UART disponíveis no sistema. O ficheiro *uart.c* contém as configurações e funções para o envio de dados pela UART. Por último, o ficheiro *XMODEM.c* contém a implementação do protocolo XMODEM para a deteção de erros e transferência de ficheiros pela porta série de ficheiros enviados pelo computador.

O *firmware* disponibilizado pelo fabricante tem uma *baudrate* definida de 2,5 Mbps. Normalmente os conversores USB -> RS232 não permitem velocidades tão elevadas. Para resolver esta situação foi utilizado um microprocessador UNO32 (programado e fornecido pelo orientador desta dissertação) que fazia de “ponte” entre o computador e a *smart camera*. A porta série deste microprocessador que comunicava com o PC trabalhava a 115200 bps (uma velocidade suportada pelo conversor USB -> RS232) e a porta série que comunicava com a *smart camera* trabalhava aos tais 2,5 Mbps. Alterou-se então a *baudrate* para um nível em que o conversor pudesse operar e transferiu-se o ficheiro *loader* para reprogramar a câmara. Assim não foi mais necessário utilizar o microprocessador Uno32 como ponte de comunicação entre o PC e a *smart-camera*.

Como nota final referir-se que o *firmware* não contém nas suas bibliotecas variáveis com vírgula flutuante, o que requer a construção de alternativas na construção de algoritmos para contornar este problema.

### 3.2.1.1 Zonas de memória

Existem zonas da memória alocadas para o armazenamento temporário das imagens capturadas pela câmara ou resultantes de processamentos realizados. Na tabela 3.1 apresentam-se as várias posições de memória alocadas para o armazenamento das imagens com o seu nome representativo e respetivo endereço. Estas zonas de memória encontram-se definidas no ficheiro *srv.h*. Para além destas zonas de memórias apresentadas existem outras com vários propósitos que podem ser utilizados em contextos diferentes dos referidos pelo fabricante. Existem zonas de memória criadas para servirem de *buffers* para as comunicações *http*, estes *buffers* podem ser utilizados para guardar valores, parâmetros de retas, subimagens, etc. Os endereços de memória definidos para guardarem imagens podem também ser utilizados para armazenar outros tipos de dados consoante as necessidades dos algoritmos a serem desenvolvidos, tendo sempre em conta se o tamanho do *buffer* é suficiente para armazenar os dados que se pretende guardar.

Tabela 3. 1 - Principais zonas da memória para as imagens.

| Nome       | Endereço   | Descrição                                                                                                       |
|------------|------------|-----------------------------------------------------------------------------------------------------------------|
| DMA_BUF1   | 0x01000000 | Posição inicial do primeiro <i>buffer</i> onde a câmara por acesso direto à memória escreve a imagem capturada. |
| DMA_BUF2   | 0x01280000 | Posição inicial do segundo <i>buffer</i> onde a câmara por acesso direto à memória escreve a imagem.            |
| FRAME_BUF  | 0x01500000 | <i>Buffer</i> utilizado para armazenar imagens processadas e/ou capturadas                                      |
| FRAME_BUF2 | 0x01780000 | Semelhante ao FRAME_BUF.                                                                                        |
| FRAME_BUF3 | 0x01A00000 | Semelhante ao FRAME_BUF.                                                                                        |
| FRAME_BUF4 | 0x01C80000 | Semelhante ao FRAME_BUF.                                                                                        |
| JPEG_BUF   | 0x00F00000 | <i>Buffer</i> utilizado para armazenar imagens que resultam do método compressão JPEG.                          |

### 3.2.1.2 Captura de imagem

Nesta secção serão descritas as rotinas existentes no *firmware* que permitem fazer a captura e transferência da imagem para os diferentes *buffers*.

Sempre que se pretende capturar uma nova imagem da câmara chama-se a função: **void grab\_frame** (). Apresenta-se agora o código da mesma.

```
void grab_frame () {
    unsigned int vect[16]; //utilizado para processamentos de imagem
    int slope, intercept; //utilizado para processamentos de imagem

    unsigned int ix, ii;
    move_image((unsigned char *)DMA_BUF1, (unsigned char *)DMA_BUF2,
               (unsigned char *)FRAME_BUF, imgWidth, imgHeight);
}
```

Estão declarados alguns dados tais como *vect[16]*, *slope*, *intercept*, etc. Estes dados são usados em pequenas rotinas de processamento de imagem já existentes no *firmware*.

A função *grab\_frame()* chama a rotina *move\_image(.....)* que irá efetivamente transferir a imagem capturada pela câmara para um *buffer* desejado. Apresenta-se agora uma breve explicação dos argumentos da função *move\_image*:

- `(unsigned char *)DMA_BUF1` - É um apontador para a primeira zona de memória onde a câmara escreve a imagem.
- `(unsigned char *)DMA_BUF2` - É um apontador para a primeira zona de memória onde a câmara escreve a imagem.
- `(unsigned char *)FRAME_BUF` - Apontador para a zona de memória onde pretendemos transferir a imagem capturada.
- `imgWidth` - Inteiro que representa a largura da imagem em número de pixels.
- `imgHeight` - Inteiro que representa a altura da imagem em número de pixels.

Passa-se agora a uma explicação do funcionamento da função `move_image`.

```
void move_image(unsigned char *src1, unsigned char *src2, unsigned
char *dst, unsigned int width, unsigned int height)
{
    unsigned char *src;
    unsigned short *isrc, *idst;
    int ix;

    if (*pDMA0_CURR_ADDR < (void *)src2)
        src = src2;
    else
        src = src1;

    isrc = (unsigned short *)src;
    idst = (unsigned short *)dst;
    for (ix = 0; ix < (width * height); ix++)
        *idst++ = *isrc++;
}
```

Os argumentos da rotina já foram anteriormente explicados.

A câmara está constantemente a escrever a nova imagem em dois *buffers* diferentes por acesso direto à memória (DMA). Ela é capaz de produzir cerca de 60 fps o que quer dizer que escreve 60 vezes por segundo nestes *buffers*, 30 em cada um deles.

Relativamente ao funcionamento desta função, primeiramente é verificado se a câmara está a escrever na zona de memória 1 (DMA\_BUF1) ou na zona de memória 2 (DMA\_BUF2) com a condição *if-else*.

O apontador “`*pDMA0_CURR_ADDR`” aponta para a atual zona de memória onde a câmara está a escrever. [37]

É feito o teste se o endereço de memória onde a câmara está a escrever é ou não maior que o primeiro endereço da zona de memória “`src2`”. Se “`*pDMA0_CURR_ADDR`” for menor significa que esta a câmara estava a escrever no *buffer 1* e então deveremos ler o *buffer 2*, se for maior ou igual a câmara está a escrever no *buffer 2* e deve-se ler o *buffer 1*. O objetivo é ler o buffer onde a câmara não está a escrever o que significa que temos uma imagem integra. Seguidamente é feita uma transferência do conteúdo do *buffer* para a zona de memória desejada. De notar que os apontadores para a transferência de dados entre os *buffers* são do tipo *unsigned short*, que são um tipo de dados que ocupam quatro bytes. Isto deve-se ao facto de que a arquitetura do processador torna mais rápida a transferência de dados com 4 bytes de cada vez, e assim conseguimos um processamento mais rápido e eficiente.

A partir deste momento e tendo a imagem na zona de memória desejada já se pode efetuar as transmissões e processamentos pretendidos.

### 3.2.1.3 Formato de imagem utilizado

A câmara escreve as imagens na memória no formato YUV422 utilizando o espaço de cores YUV descrito na subsecção 2.2.1 do capítulo do estado da arte. Este formato organiza a informação relativa aos pixéis da imagem em conjuntos de quatro bytes [U Y1 V Y2]. Cada conjunto deste tipo representa dois pixéis seguidos na horizontal, a componente U e V é igual para os dois pixéis apenas variando o Y. Na primeira posição de memória do *buffer* encontra-se o conjunto de dados respetivo aos dois primeiros *pixels* do canto superior esquerdo da imagem e sequencialmente na última posição da memória encontra-se o conjunto de dados correspondente ao último par de pixéis no canto inferior direito da imagem.

Cada imagem tem um número de altura de pixéis (*imgHeight*) e um número de largura (*imgWidth*).

Cada conjunto de dados representa 2 pixéis e tem o tamanho de 4 bytes, 1 byte para cada componente U, Y1, V, Y2.

Então o tamanho total de bytes (*n\_bytes*) necessário para armazenar uma imagem de resolução *imgWidth* x *imgHeight* é dado por:

$$n_{bytes} = 4 * \frac{imgWidth * imgHeight}{2} = 2 * imgWidth * imgHeight \quad (3.1)$$

### 3.2.1.4 Descrição dos processamentos de imagem existentes no *firmware*

O *firmware* traz já alguns processamentos de imagem *standard* implementados para aplicações robóticas.

- ***frame differencing*** - Guarda na memória uma imagem de referência, e é a partir daí que todas as novas imagens capturadas serão subtraídas à imagem de referência pelas seguintes fórmulas:

$$Y = Y_n - Y_{ref} \quad (3.2)$$

$$U = U_n + 128 - U_{ref} \quad (3.3)$$

$$V = V_n + 128 - V_{ref} \quad (3.4)$$

Em que  $Y_n$ ,  $U_n$  e  $V_n$  são os valores da imagem capturada e  $Y_{ref}$ ,  $U_{ref}$  e  $V_{ref}$  são só valores da imagem de referência,  $Y$ ,  $U$  e  $V$  são os valores dos pixéis que formam a imagem resultante deste processo.

- ***Blob*** - Através do comando 'vb' é feita uma pesquisa na imagem por *blobs* que estejam compreendidos numa gama de cores previamente definida. As respostas dadas pelo sistema têm o seguinte formato: "###vbc dd\r\n ssss x1 x2 y1 y2\r\n ...", em que *dd* é o número *blobs* encontrados, *ssss* é o número de pixéis existentes no

blob e x1, x2, y1 e y2 definem a o retângulo que contem o *blob*. Podem ser detetados até 16 *blobs*.

- **Horizon detection** - Depois de enviado o comando “g3”, o sistema começa à procura de variações horizontais de cor e luminância para detetar a linha do horizonte. Deteta apenas retas horizontais e apenas uma por imagem. Quando as imagens são enviadas para o *host* a reta detetada é imprimida na imagem.
- **Edge detection** - Através do comando “g2” faz-se um processamento de imagem básico para a marcação de arestas na imagem. Este processamento é bastante rápido, no entanto, da aplicação deste algoritmo resultam arestas com vários pixéis de largura pelo que para haver um processamento posterior poderia ser necessário algoritmos de erosão.
- **Histogram** - Com o comando “vh” é calculada uma lista de distribuição dos valores de Y,U e V da imagem em intervalos de 3 em 3 começando no 0 e acabando em 255. Não é precisa uma aplicação específica para visualizar este histograma porque o histograma é enviado em forma de texto e não em forma gráfica.
- **Pixel Sample** - Para saber os valores absolutos de um determinado pixel da imagem deve ser enviado o seguinte comando: “vpxxxxyyyy”, em que **xxxx** e **yyyy** descrevem as coordenadas do pixel a amostrar.

### 3.3 Modos de interação com o sistema

São três os modos principais de interação com a *smart camera*. Existem várias formas de comunicar com o sistema, desde consolas especialmente desenhadas para esta aplicação, através do *browser* da internet, *Labview*, *Matlab*, *smartphones*, consolas de jogos, etc . No entanto, por trás destas formas de comunicação estão os três modos principais de interação. De seguida far-se-á a sua descrição.

#### 3.3.1 Primeiro modo de interação

É o modo mais simples e o mais limitado também. O sistema está já programado com um *firmware* fornecido pelo fabricante. A ideia básica deste modo de funcionamento é o envio de comandos para o sistema consoante as necessidades do *host*. O *firmware* está desenhado de acordo com o protocolo SRV que é o protocolo que rege toda a comunicação feita com o sistema. Este primeiro modo de funcionamento possibilita a transferência de um ficheiro *loader* com o *firmware* compilado para o sistema e de programa-lo.

O principal comando deste modo de funcionamento é o utilizado para atualizar o *firmware*. Antes de atualizar o *firmware* o sistema verifica se o *firmware* existente no *flash buffer* que se pretende programar no sistema é válido. Contudo este método não é infalível e poderá ser instalado um *firmware* não válido, ou seja, o *bootloader* poderá estar danificado,

o que nos impedirá de atualizar o *firmware* posteriormente. Em certos casos poderá ser necessário um dispositivo programador com interface JTAG.

Como a informação sobre o processo de atualização do *firmware* está muito dispersa pelo fórum do fabricante e por algumas páginas criadas pelos usuários do sistema foram criados tutoriais acerca da compilação e atualização do *firmware* em sistemas operativos Windows e Linux. Estes tutoriais encontram-se no fim deste documento na secção de Anexos.

Outro comando com elevada importância é o que nos permite transferir o código em PicoC para o interpretador de C existente no *firmware* disponibilizado pelo fabricante.

Para além dos comandos apresentados anteriormente existem outros com interesse neste modo de interação como por exemplo: comandos de captura e envio de imagens para o *host*, comandos de processamento de imagem descritos na subsecção 3.2.1.4, comandos de configuração de parâmetros como nível de qualidade de imagem na compressão JPEG, resolução de imagem, etc.

Embora para a atualização do *firmware* seja necessário um PC, para os demais comandos podem ser utilizadas várias plataformas que possibilitarão a comunicação com a *smart camera* desde que tenham aplicações desenvolvidas para este fim. O fabricante dispõe no seu sítio da internet de várias aplicações que permitem comunicar com a câmara através de *smartphones* e consolas de jogos. As comunicações nestes casos são feitas através da rede *wireless*.

Todos os comandos disponíveis no protocolo SRV encontram-se no *site* oficial do fabricante.

### 3.3.2 Segundo modo de interação

O *firmware* possui um interpretador de C. Este é baseado no interpretador PicoC Open Source. Através de pequenos programas escritos em PicoC é possível obter comportamentos desejados do sistema. Os programas são enviados para o *flash buffer* pelo comando 'X' através do protocolo XMODEM e depois executados pelo interpretador de C. Através de sequências de funções poderemos obter um comportamento do programa diferente do standard oferecido pelo protocolo SRV. O interpretador não suporta todas as funcionalidades oferecidas pela ISO C. O fabricante disponibiliza uma lista das funções, declarações, atribuições e procedimentos suportados.

### 3.3.3 Terceiro modo de interação

Consiste na alteração/criação do *firmware* fornecido pelo fabricante da câmara de modo a satisfazer as necessidades da aplicação em que o sistema será empregue. Desta forma, obtemos um sistema com respostas mais personalizadas, e adequadas às necessidades de interface e processamento de imagem dos restantes sistemas que integram a aplicação robótica. Comparativamente ao segundo modo de interação este modo apresenta uma lista de funcionalidades da linguagem C completa de acordo com a ISO C e oferece um processamento mais rápido das instruções.

O fabricante disponibiliza duas versões do *firmware* distintas. Uma versão construída de raiz para este sistema e outra baseada no uClinux, sistema operativo para microcontroladores. O trabalho será desenvolvido na primeira das versões.

Para efetuar alterações ao *firmware* pesquisou-se uma forma de integrar a *Blackfin toolchain* num ambiente de desenvolvimento integrado (IDE). O IDE escolhido foi o Eclipse. O Eclipse é um ambiente de desenvolvimento integrado escrito em *java* e desenvolvido inicialmente pela empresa IBM. Trata-se de um *software* open source baseado no desenvolvimento de *Plug-ins* para atualizar e adicionar funcionalidades ao programa.

Durante a pesquisa do processo para integrar a *toolchain* no eclipse verificou-se que a informação encontrava-se muito dispersa, desatualizada e com muitos *links* inexistentes. Decidiu-se então construir um tutorial que agregasse toda esa informação necessária para a integração. Este tutorial encontra-se no fim deste documento na secção de anexos.



## Capítulo 4

### Descrição do sistema AGV

Este capítulo tem como objetivo informar o leitor acerca dos *Landmarks* a serem detetados e do sistema onde estes se inserem. Será feita uma descrição do sistema como um todo e depois uma descrição mais pormenorizada dos *Landmarks*.

Esta deteção insere-se nos trabalhos desenvolvidos pelos estudantes David Lima [38] e Roberto Silva [4] no âmbito da dissertação de mestrado do MIEEC.

Trata-se de um sistema de localização de AGVs baseado em marcadores. Este capítulo é resumidamente uma descrição do trabalho feito pelos dois estudantes anteriormente referidos.

#### 4.1 Apresentação geral do AGV

O robô onde se encontra todo o sistema implementado está representado na figura seguinte.

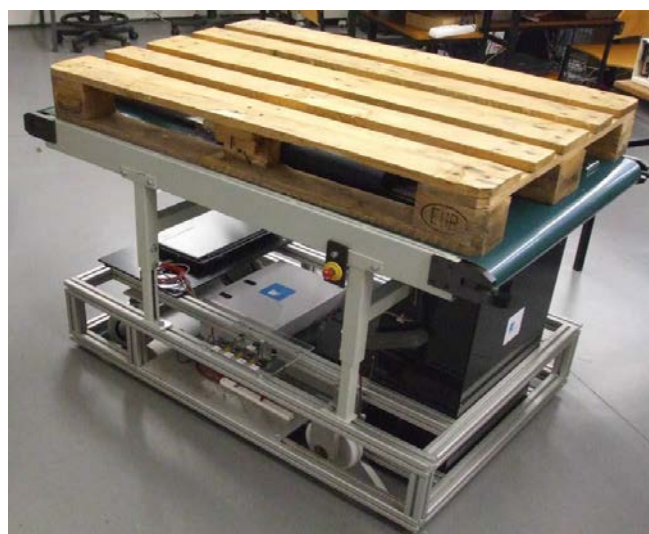


Figura 4.1- Robô móvel industrial. [8]

Mecanicamente este robô foi projetado para efetuar o transporte dos mais diversos materiais da indústria através das paletes que podem ser colocadas sobre ele. Está dimensionado de acordo com a norma europeia do transporte de paletes (800\*1200\*144 mm).

Em traços gerais o robô é constituído por:

- Um tapete rolante para cargas e descargas de material;
- Um par de rodas com sistema de tração diferencial na parte da frente e um par de rodas castor na traseira;
- Um sistema de visão artificial para detetar marcadores na parte frontal;
- Uma caixa com proteção contra interferências eletromagnéticas existentes em ambientes industriais, onde se encontra o sistema elétrico embutido;
- Duas baterias de chumbo de elevada capacidade de carga;
- Uma plataforma onde pode ser colocado um computador portátil.

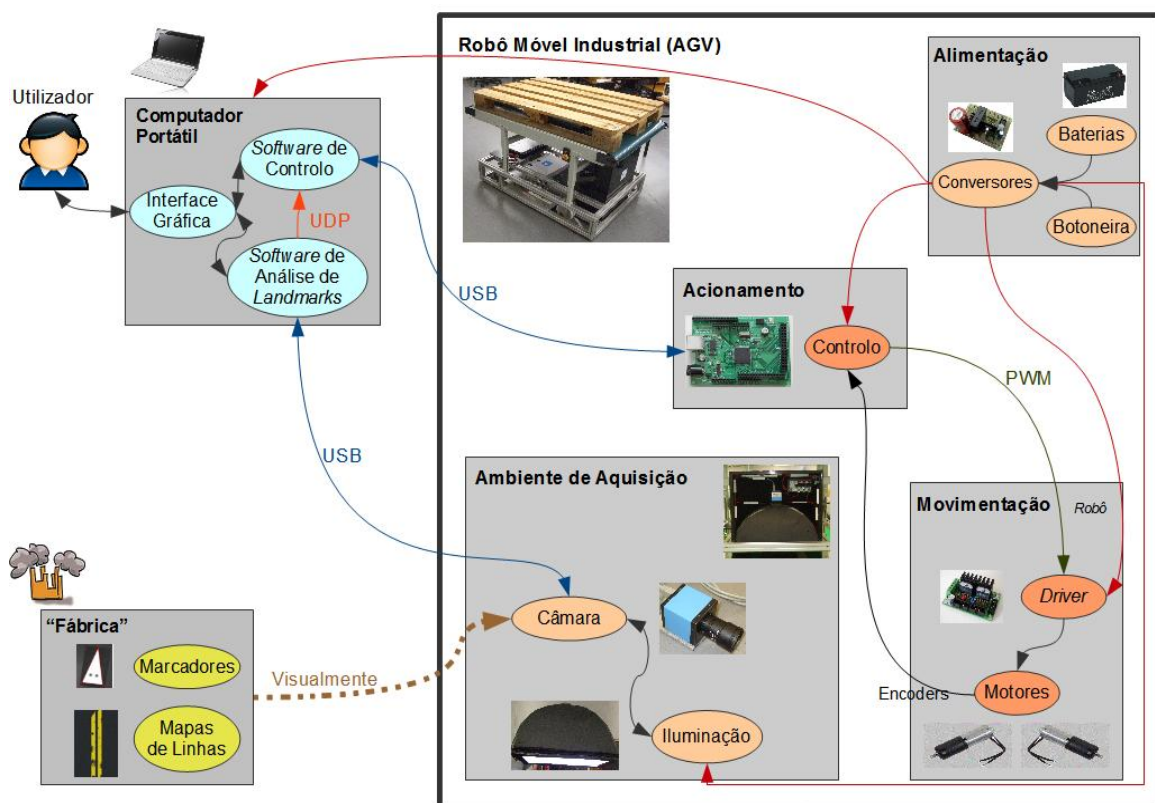


Figura 4.2 - Diagrama funcional do sistema. [8]

Através do diagrama acima apresentado é possível ter uma ideia geral da arquitetura funcional do sistema.

O computador portátil é responsável por correr o *software* de alto nível que faz as tarefas de interface gráfica e controlo de alto nível. O *software* de baixo nível é executado no microcontrolador e é responsável pelo controlo dos motores consoante as ordens dados pelo sistema de controlo superior (computador).

O *software* de controlo recebe periodicamente informação sobre os *encoders*, os dados resultantes do pós processamento de imagens efetuado pelo *software de análise* de imagem, e os parâmetros indicados pelo utilizador. A partir destes dados são tomadas decisões e

enviadas informações para o microcontrolador para que este acione os motores consoante as ordens dadas.

Visto que a descrição do sistema elétrico do robô não tem interesse no âmbito da atual dissertação essa descrição não será feita. Será dada uma maior ênfase ao sistema de visão artificial constituído pelo ambiente de aquisição, ao *software* de análise de imagem e descrição dos marcadores. De referir que um dos objetivos da presente tese é efetuar o processamento do *software* de análise de imagem na *smart camera* em estudo.

## 4.2 Sistema de visão artificial

O sistema de visão artificial foi concebido para que fosse o mais robusto possível de forma a evitar as interferências externas. Para tal, foi construída uma caixa onde estão incorporados todos os componentes deste sistema de visão: câmara digital, iluminação controlada, sistema de ventilação e computador portátil para análise das imagens. Esta caixa encontra-se na parte frontal do robô como exemplifica a figura seguinte.

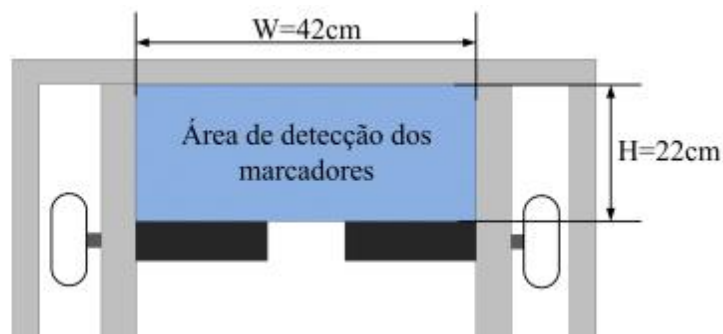


Figura 4.3 - Dimensões e posição da zona de aquisição de imagem. [45]

A câmara digital é uma DMK 31AU03\_[38] fabricada pela empresa *The Imaging Source*.



Figura 4. 4 - Câmara DMK 31AU03. [4]

Especificações:

- **Resolução:** 1024x768, monocromática;
- **Comunicação e Alimentação:** USB;
- **FPS máxima:** 30;
- **Consumo Típico:** 2500mW;
- **Preço:** 490.00€;
- **Lente:** LTC 3361/30;

### 4.3 Plataformas do Software

Existem três aplicações principais: a de análise de informação das imagens, a de controlo e localização do AGV, e a de comunicação com o microcontrolador existente no robô. Todas estas aplicações foram desenvolvidas na plataforma Lazarus, que é um ambiente de desenvolvimento *opensource* baseado em *FreePascal*.



Figura 4.5 -Interface gráfica da aplicação de análise de imagens. [8]

Esta aplicação tem como principal função analisar as imagens e retirar destas as informações presentes nos marcadores e posteriormente envia-las para a aplicação de controlo e localização. Existem também uma série de funções adicionais que permitem ao utilizador a configuração dos parâmetros da câmara e alterar os parâmetros dos vários algoritmos implementados.

A aplicação de controlo e localização do AGV [39] foi adaptada de um projeto anterior chamado *RoboVigil*.

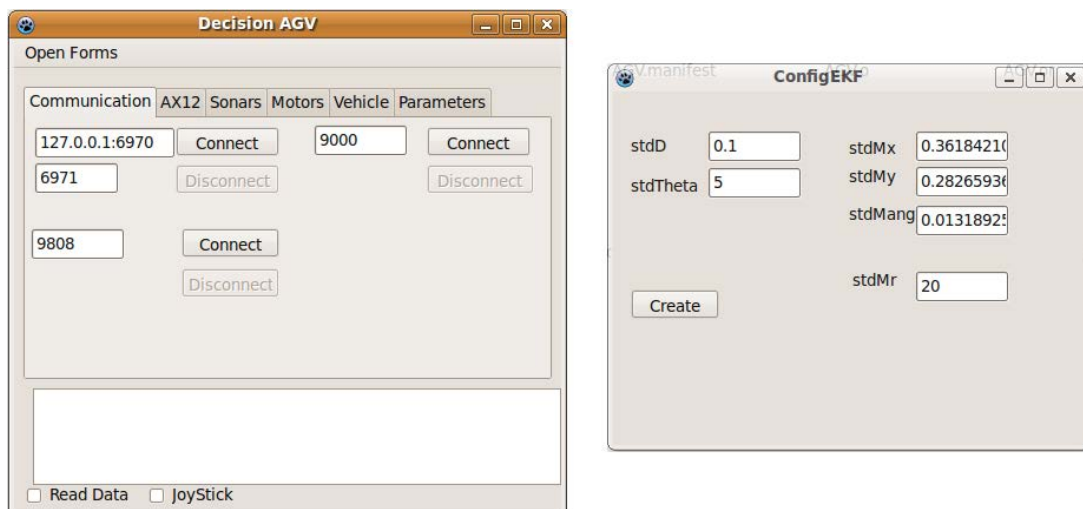


Figura 4.6 - Interface gráfica do sistema de controlo. [8]

À esquerda é apresentada a interface do sistema de controlo e à direita a interface do sistema de localização. Esta aplicação com base nos dados recebidos pela aplicação de

análise de imagens faz uma estudo aos mesmos e decide qual a movimentação que o AGV deve fazer. Depois comunica essa mesma decisão à aplicação que efetua a comunicação com o microcontrolador. Para além disso e como na aplicação anterior é ainda possível definir os vários parâmetros dos diferentes algoritmos.

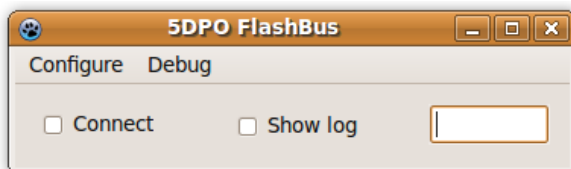


Figura 4.7 - Interface da aplicação de comunicação com o microcontrolador. [8]

Esta aplicação foi desenvolvida pela equipa 5DPO [40], e tem como principal objetivo converter o tipo de comunicação de entrada num tipo de comunicação que seja suportado pelo microcontrolador e vice-versa. O código que é executado no microcontrolador (Megavore) também foi desenvolvido no âmbito do projeto RobVigil.

A aplicação de controlo e localização, a aplicação de comunicação e a aplicação de análise de imagens comunicam entre si utilizando o protocolo UDP, que é um protocolo muito utilizado em sistemas de tempo real. A comunicação entre a aplicação de comunicação e o microcontrolador é feita através do protocolo USB.

A trama enviada da aplicação de análise de imagem para a aplicação de controlo tem o seguinte formato:

Tabela 4.1 - Formato da trama resultante do processamento de imagem. [45]

| Tipo de Landmark | Posição absoluta | Posição relativa | Ângulo medido | Assinatura | Fim |
|------------------|------------------|------------------|---------------|------------|-----|
| 'A' ou 'L'       | Xa,Ya            | Xm,Ym            | $\theta_m$    | 'C'        | 'E' |

## 4.4 Descrição dos *Landmarks*

Os *landmarks* encontram-se dispostos no solo para que o robô os possa capturar, analisar e tomar decisões com base nas características extraídas. São os *landmarks* que permitem ao robô identificar a sua localização e saber para onde tem de ir no futuro imediato. O objetivo desta secção é apresentar os *landmarks* resultantes dos dois trabalhos anteriores já referidos e que serão muito importantes no âmbito da atual dissertação, uma vez que pretende-se desenvolver e implementar algoritmos na *smart camera* que sejam capazes de descodificar estes mesmos *landmarks*.

Os *landmarks* são figuras geométricas que se pretende capturar e identificar e podem ser linhas e marcadores. Primeiramente faz-se uma descrição dos marcadores.

Os marcadores que foram escolhidos possuem a forma de um triângulo. A configuração utilizada é apresentada na figura seguinte.

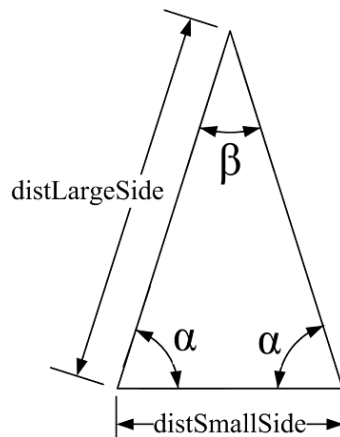


Figura 4.8 - Forma do marcador desenvolvido. [45]

Tabela 4.2 - Dimensões do marcador. [45]

|                         |         |
|-------------------------|---------|
| distLargeSide           | 10cm    |
| distSmallSide           | 4,72 cm |
| Ângulo de base $\beta$  | 27,2°   |
| Ângulo de base $\alpha$ | 76.4°   |

A forma deste marcador permite saber facilmente o seu sentido e direção devido ao seu formato de seta. Isto origina um processamento eficiente visto que a matemática necessária para extrair as suas características não é muito complexa.

Inicialmente estes marcadores eram feitos de papel e com o auxílio de uma fita adesiva transparente permitiam a sua colocação no chão. Como esta é uma solução que não se adapta bem ao meio industrial foram recriados os mesmos marcadores em plástico.

Para além da informação de direção e sentido os marcadores também têm de fornecer instruções acerca da localização absoluta atual do robô. Para isso estes são codificados. Dentro de cada triângulo existem zonas identificadas que permitem fazer uma codificação como exemplifica a figura seguinte.

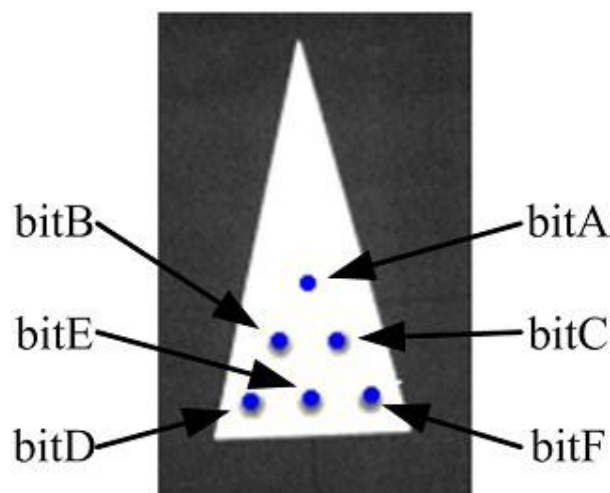


Figura 4.9 - Codificação dos marcadores. [45]

O cálculo do código de cada posição é feito da seguinte forma:

$$\text{Código} = 2^0 * \text{bitA} + 2^1 * \text{bitB} + 2^2 * \text{bitC} + 2^3 * \text{bitD} + 2^4 * \text{bitE} + 2^5 * \text{bitF} \quad (4.1)$$

O robô possui uma base de dados que permite associar cada código marcado a uma posição no espaço.

Relativamente às linhas, são *landmarks* em forma de retângulo e fornecem informação apenas sobre a direção da trajetória pretendida, essa direção é definida pelas arestas mais longas do retângulo. As linhas servem para eliminar os desvios que possam ocorrer enquanto o robô se encontra a ir de um marcador para outro. Este tipo de *landmark* pode também ser uma medida de segurança em caso de perda total do robô, uma vez que o mantém dentro de uma trajetória aceitável evitando embates contra bens e pessoas. Estas linhas são feitas de fita adesiva visto não ser necessário ter grandes cuidados com este tipo de marcadores. O essencial é a existência de um grande contraste entre a fita e o chão.

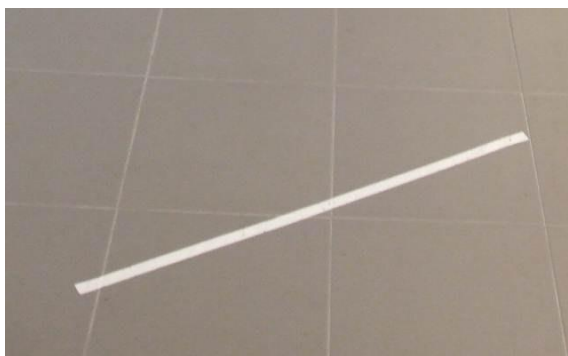


Figura 4.10 - Linha utilizada. [8]

## 4.5 Definição dos *outputs* da *smart camera*

Um dos objetivos da atual dissertação é passar o processamento que é feito pela aplicação de análise de imagens para a *smart camera*. Assim serão construídos de raiz os vários algoritmos de processamento para serem processados pelo sistema em estudo.

Tendo em conta o diagrama funcional presente na figura 4.2 o objetivo passa por substituir a câmara DMK 31AU03 pelo sistema em estudo nesta dissertação. Pretende-se também passar o processamento efetuado na aplicação de análise de imagens para o processamento *on board* da Surveyor Blackfin Camera. A *smart camera* passará então a comunicar diretamente com a aplicação de controlo e localização e deverá fornecer a esta os dados existentes na trama descrita na tabela 4.1.



# Capítulo 5

## *Software* desenvolvido

Este capítulo tem como objetivo apresentar o *software* desenvolvido no âmbito desta dissertação. O software desenvolvido pode ser dividido em duas partes principais: 1- A aplicação desenvolvida na plataforma *Lazarus* para interação com a *smart camera* através do computador, esta comunicação faz-se através da porta série, e 2- O código desenvolvido para a implementação de algoritmos de imagem a serem executados no sistema em estudo.

*Lazarus* é um ambiente de desenvolvimento integrado *opensource* e baseado no compilador FreePascal.

Começa-se por apresentar a aplicação desenvolvida para interagir com a câmara.

### 5.1 Aplicação de interação com a câmara

Esta aplicação foi construída de raiz no âmbito desta dissertação para proporcionar um meio de interação com o sistema em estudo. Numa primeira fase serviu para testar os comandos do protocolo SRV e depois foi uma preciosa ajuda no *debugging* dos algoritmos de processamento de imagem desenvolvidos.

#### 5.1.1 Funcionalidade desenvolvidas

Nesta aplicação desenvolveram-se as seguintes funcionalidades:

- Receção e armazenamento das imagens capturadas e/ou resultantes de processamentos de imagem. As imagens podem ser recebidas em formato JPEG ou em formato BMP. O *firmware* disponibilizado pelo fabricante já possuía uma rotina de envio de imagens em formato JPEG para o *host*, no entanto, como se pretendia receber imagens sem perda de informação foi desenvolvida uma rotina que envia as imagens no formato BMP.
- Interação com o sistema utilizando o protocolo SRV anteriormente descrito.

- Receção dos tempos de processamento dos algoritmos implementados.
- Envio de imagens em formato BMP para a *smart camera* para posterior processamento *onboard*.
- Alteração dos diversos parâmetros dos algoritmos desenvolvidos.

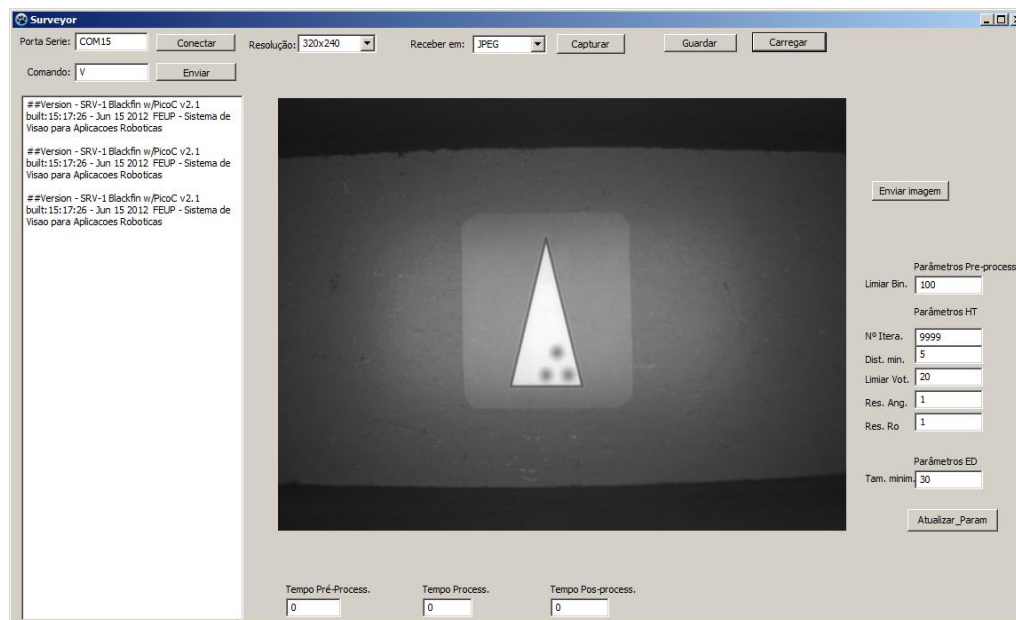


Figura 5.1 - Aspeto da aplicação desenvolvida em Lazarus.

### 5.1.2 Rotinas desenvolvidas na *smart camera* para interação com a aplicação

Para a criação das funcionalidades acima referidas foi necessário desenvolver algumas rotinas no sistema em estudo por forma a haver uma comunicação compatível entre a aplicação (*host*) e a *smart camera*. Nesta secção serão explicados o funcionamento e a estrutura destas rotinas desenvolvidas. Foi necessário construir as seguintes rotinas: conversão de imagem para o formato BMP e envio pela porta série para o *host*, receção da imagem em formato BMP e conversão para o formato YUV, medição de tempos de processamento e atualização dos parâmetros das funções de processamento de imagem. As restantes funcionalidades do protocolo SRV já estavam implementadas no *firmware* de origem.

#### 5.1.2.1 Conversão da imagem para o formato BMP

Pretendia-se receber imagens sem qualquer tipo de compressão para serem objeto de uma análise visual detalhada ao nível de pixel a pixel após as imagens serem submetidas a determinados processamentos. Já existia uma rotina que permitia receber as imagens em formato JPEG, contudo estas apresentavam compressão e não satisfaziam os requisitos.

Como explicado na subsecção 3.2.1.3 as imagens capturadas na memória RAM encontram-se no formato YUV, assim foi necessário implementar a conversão do espaço YUV para o RGB.

Primeiramente construiu-se uma rotina de conversão e numa segunda fase uma rotina que adiciona o *header bitmap* e envia a imagem pela porta série.

Na subsecção 2.2.1.2 foram apresentadas as fórmulas para esta conversão, mas estas não puderam ser utilizadas visto que as bibliotecas existentes não suportam números com virgula flutuante. Assim foi necessário realizar uma nova pesquisa na literatura de forma a contornar o problema. Foi encontrado o seguinte código [41] que apresentou resultados satisfatórios.

```
for(yy = 0; yy < imgHeight; yy++) {
    for(xx = 0; xx < imgWidth; xx += 2) {
        idxFrame = index(xx, yy); //calcula a posicao
        // parte comum para os dois pixels
        u = frameBuffer[idxFrame]; //Componente U
        v = frameBuffer[idxFrame + 2]; // Componente V
        // YUV -> RGB
        u -= 128; //
        v -= 128;
        v1 = (5727 * v);
        uv1 = -(1617 * u) - (2378 * v);
        u1 = (8324 * u);
        // Pixel #1
        y = frameBuffer[idxFrame + 1];
        y1 = y << 12;
        r = (y1 + v1) >> 12;
        g = (y1 + uv1) >> 12;
        b = (y1 + u1) >> 12;
        if (r > 255) r = 255;
        else if (r < 0) r = 0;
        if (g > 255) g = 255;
        else if (g < 0) g = 0;
        if (b > 255) b = 255;
        else if (b < 0) b = 0;
```

Após a conversão estar feita é então necessário criar o *header* do formato BMP de forma que a aplicação Lazarus consiga decodificar os resultados. Este *header* foi detalhadamente descrito na subsecção 2.2.2.1. Na construção de uma rotina que criasse o *header* BMP ocorreram muitos erros, e tardava em conseguir um *header* não corrompido, ou seja, o ficheiro era transferido para o *host*, era guardado mas não era reconhecido como uma imagem. Para resolver este problema pesquisou-se um programa que permite-se verificar o conteúdo de ficheiros binários. Foi encontrado o programa *Hex Editor XVI32* [42], que permite visualizar e alterar o conteúdo de ficheiros binários em formato hexadecimal. Este ficheiro foi também importante para se poder fazer para além de uma análise visual também uma análise analítica aos processamentos de imagem através da verificação dos valores de cada pixel.

### 5.1.2.2 Conversão da imagem do formato BMP para o formato YUV

A necessidade da criação desta rotina prende-se com o facto de ser necessário testar os algoritmos implementados primeiramente com imagens virtuais (imagens criadas no computador) antes do teste com imagens reais. Como os processamentos de imagem implementados foram construídos para trabalhar com o formato YUV que é o formato em que

a câmara escreve na memória RAM é necessário converter as imagens virtuais criadas no computador para este formato. Foram encontradas várias fórmulas para esta conversão mas novamente o problema da falta suporte de números com vírgula flutuante imponha-se. Após uma pesquisa mais intensa encontrou-se as seguintes fórmulas [43] que foram usadas na conversão.

$$Y1 = ( ( 66 * R + 129 * G + 25 * B + 128) >> 8) + 16; \quad (5.1)$$

$$U1 = ( ( -38 * R - 74 * G + 112 * B + 128) >> 8) + 128; \quad (5.2)$$

$$V1 = ( ( 112 * R - 94 * G - 18 * B + 128) >> 8) + 128; \quad (5.3)$$

### 5.1.2.3 Medição de tempos de processamento

Um fator a ter em conta muito importante é o tempo que um algoritmo leva a ser processado visto que estamos a falar de uma aplicação em tempo real. Para isso foi necessário calcular os tempos de processamento para este depois serem enviados para o *host* e poderem ser apresentados na aplicação.

A medida dos tempos foi algo muito fácil de implementar visto que no *firmware* original existe uma variável que é incrementada a cada milissegundo passado desde o último *reset* efetuado ao sistema. Assim antes da execução de um algoritmo o valor dessa variável é lido e depois da execução do algoritmo é lida novamente a variável em questão e a esse valor subtraído o valor que a variável tinha antes do início do processamento. Com isto consegue-se obter uma precisão ao milissegundo.

### 5.1.2.4 Atualização dos parâmetros dos vários processamentos

Para a atualização dos parâmetros são enviados por parte do *host* os valores introduzidos pelo utilizador. Depois estes valores atualizam as diversas variáveis que representam os parâmetros de cada processamento.

## 5.2 Processamentos de imagem implementados

Os algoritmos de processamento de imagem a serem implementados têm como objetivo a identificação dos *Landmarks* referidos no capítulo 4. Pretendeu-se implementar um algoritmo de extração das retas existentes na imagem para um posterior processamento que permita a identificação dos marcadores.

Inicialmente foi feita uma pesquisa pelos algoritmos existentes na literatura que permitissem obter bons resultados tendo sempre em conta a duração temporal dos mesmos. Foi primeiramente escolhida a transformada de Hough aleatória para ser implementada. Sempre com o objetivo de obter um processamento o mais leve possível, depois de ser implementada a transformada de Hough aleatório foi feita uma nova pesquisa bibliográfica e encontrada uma nova técnica de extração de linhas muito recente chamada EDlines. Nesta secção do presente capítulo irá ser feita uma descrição pormenorizada da implementação dos algoritmos acima referidos. Na implementação destes algoritmos não se limitou apenas a

seguir as técnicas dos autores, foram também realizadas algumas melhorias dos algoritmos para este caso específico com o objetivo de minorar o tempo de processamento.

Depois de descritos os algoritmos de processamento implementados será ainda explicado o pós-processamento implementado.

### 5.2.1 Transformada de Hough Aleatória

Esta variante da transformada de Hough [28] caracteriza-se por ser mais leve em termos de processamento que a transformada original, devido à sua essência probabilística. A principal poupança em termos de tempo está no facto da seleção de alguns pontos aleatórios para o cálculo da reta correspondente, enquanto na versão original eram utilizados todos os pontos de informação (pixéis diferentes do fundo) da imagem. Por outras palavras a transformada de Hough Aleatória obtém a partir dos pixéis de informação de uma imagem binária, um pequeno subconjunto de pontos selecionados aleatoriamente, a partir dos quais se faz um mapeamento para o novo espaço de parametrização da curva procurada, neste caso a reta.

O método está baseado no fato de que um ponto no novo espaço de parametrização pode ser determinado a partir de um pequeno subconjunto de pontos de uma aresta da imagem original, como ilustra a figura 5.2. O tamanho deste subconjunto depende exclusivamente da complexidade das curvas a serem detetadas. Neste caso particular, o caso das retas, o ponto de acumulação no novo espaço de parametrização pode ser obtido a partir de apenas dois pontos.

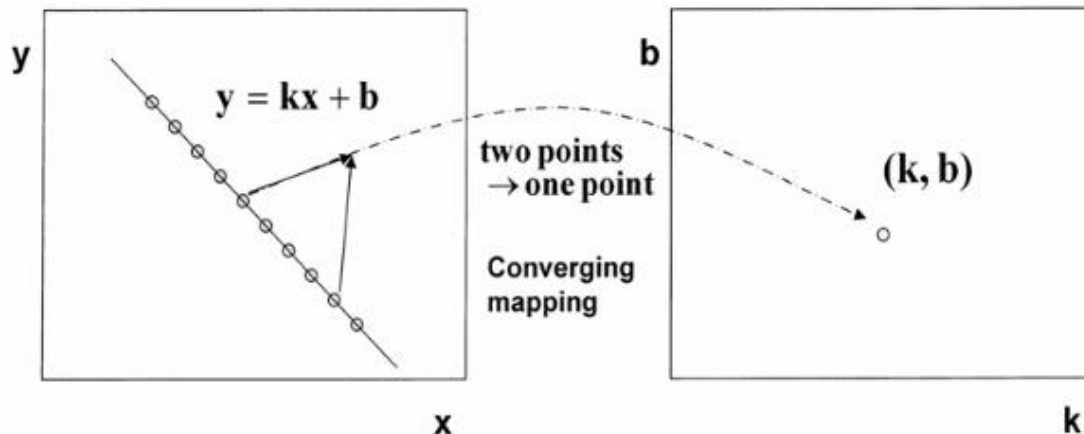


Figura 5.2 - Transformação entre espaços da Transformada de Hough Aleatória. [34]

As vantagens principais desta variante em relação à versão clássica são as seguintes:

- Implementação relativamente fácil.
- Menos exigência em termos de processamento.
- Rácio robustez/tempo de processamento mais elevado.
- É mais adaptada a sistemas de tempo real.

Este algoritmo inicia-se com uma imagem sujeita ao processo de binarização. Este processo foi já explicado na secção 2.5.1.3, como tal, não será alvo de explicação nesta fase.

Apresenta-se agora uma sucinta explicação de todo o algoritmo. Para o caso de detecção de retas, todos os passos serão explicados em detalhe mais à frente. Existem várias variantes deste método, cada uma para aplicações diferentes. Decidiu-se apresentar o algoritmo *standard*. Mais tarde na parte de implementação, serão explicadas todas as alterações feitas.

1. Criar um conjunto (D) com todos os pixéis de informação da imagem binarizada.
2. Selecionar aleatoriamente dois pontos do conjunto D.
3. Verificar se os dois pontos estão a uma distância mínima. Caso não estejam, é necessário voltar ao passo 2.
4. Calcular os parâmetros  $\theta$  e  $r$  com os dois pontos anteriormente selecionados.
5. Ir ao acumulador e incrementar o contador da posição ( $\theta$ ,  $r$ ) que anteriormente já foram calculados. Verificar se o valor dessa posição do acumulador é superior a um limiar anteriormente definido. Se sim guardar os valores ( $\theta$ ,  $r$ ) no vetor de retas detetadas e eliminar da imagem os pontos pertencentes a essa reta.
6. Verificar se os passos de 1 a 5 foram efetuados a vezes desejadas ou se já foi detetado o número de retas desejado. Se sim acaba aqui o algoritmo, caso contrário regressar ao passo 1.

Apresenta-se agora uma descrição mais detalhada de cada passo.

**Passo 1** - É criado um vetor com as coordenadas dos pixéis de informação. Como a imagem passou já por um processo de binarização, este passo consiste em fazer um “scan” a todas as posições da memória da imagem e verificar se o pixel tem luminosidade máxima. Em caso afirmativo é guardado no conjunto de os valores de  $x$  e  $y$  correspondentes às coordenadas cartesianas do pixel. A ordem de entrada dos diferentes pixéis no conjunto D não é relevante uma vez que mais tarde estes passarão por um processo de seleção aleatória.

**Passo 2** - Esta etapa consiste na seleção aleatória de dois pixéis de informação.

**Passo 3** - É necessário verificar se os dois pixéis selecionados anteriormente não se encontram demasiado “próximos” ou demasiado “afastados”. No caso de estarem a uma distância muito curta isto pode originar uma votação falsa. Por exemplo, dois pixéis juntos na horizontal quando a reta da qual fazem parte tem uma orientação vertical. No segundo caso, para se definir uma distância máxima é necessário ter um prévio conhecimento dos tipos e dimensões das retas que se quer detetar.

**Passo 4** - Nesta fase calcula-se os parâmetros da reta que formam os dois pontos selecionados aleatoriamente na fase anterior.

De notar que são utilizadas coordenadas polares. Isto deve-se ao facto de poderem existir retas que possuem um declive infinito se fossem utilizadas coordenadas cartesianas uma vez que a variação no eixo das abcissas seria nula.

**Passo 5** - Nesta fase incrementa-se o contador na posição do acumulador com os dois parâmetros anteriormente calculados  $\theta$  e  $r$ .

A diferença dos valores  $\theta$  e  $r$  entre cada posição do acumulador é definida por resolução do acumulador.

Esta fase é crítica para o sucesso da deteção. Repara-se que se os dois pontos forem selecionados de retas diferentes ou de pontes de ruído, é muito improvável que estes formem os mesmos valores de  $\theta$  e  $r$  consecutivamente, assim sendo os contadores destas posições terão valores baixos porque raramente serão detetados repetidamente os mesmos valores de  $\theta$  e  $r$ . Isto ocorre quando os dois pontos selecionados aleatoriamente pertencem a retas diferentes e/ou a pontos de ruído. Já se os dois pontos pertencerem à mesma reta serão obtidos sempre os mesmos valores de  $\theta$  e  $r$ . É depois verificado se o contador dessa posição da memória tem um valor superior a um limiar. Se tiver considera-se que foi detetada uma reta com os valores de  $\theta$  e  $r$  dessa posição. Estes valores são introduzidos no vetor de retas detetadas e são eliminados os pixels de informação da imagem pertencentes a essa reta. Assim os pixels desta reta não entram para as próximas votações o que acelera o processo.

**Passo 6** - Para evitar possíveis ciclos infinitos é necessário definir um número máximo de iterações deste ciclo. Este número máximo de iterações depende do número máximo de retas a ser detetadas e da quantidade de ruído presente na imagem. Este valor pode ser definido através da análise experimental do comportamento do sistema. Aqui é verificado se o número máximo de iterações foi atingido ou se o número de retas que se pretende detetar foi atingido. Se sim o algoritmo acaba aqui, caso contrário regressa-se ao passo 1.

#### 5.2.1.1 Implementação da RHT

Esta *smart camera* possui uma capacidade de processamento relativamente forte, no entanto, não chega aos níveis considerados ideais. Nesta subsecção descrever-se-á o algoritmo implementado, as modificações efetuadas, as limitações existentes e as formas como estas foram contornadas. Segue-se uma descrição de todos os passos do algoritmo implementado. De salientar que são utilizadas apenas as componentes de luminância da imagem, isto porque não é necessário a diferenciação da cor nesta aplicação específica e torna o processamento mais eficiente.

##### **1. Pré-processamento**

Nesta fase é efetuado o pré-processamento à imagem, que é necessário para que haja um realce das orlas da imagem. Para este efeito poderia ser utilizado qualquer um dos operadores apresentados na subsecção 2.5.1.2. Sempre com o objetivo de obter um tempo de processamento o mais reduzido possível foi decidido utilizar o operador Canny. Esta escolha deve-se ao facto de ser apenas feita uma convolução com uma matriz. Isto reduz o tempo de processamento para metade em relação aos outros operadores acima referidos.

Aquando da primeira implementação deste algoritmo foi verificado que este tinha uma duração de 89 ms em imagens com resolução de 320x240. Este tempo foi considerado elevado e foi então decidido investigar formas de reduzir este tempo.

Inicialmente o cálculo do valor de cada pixel era feito da seguinte forma:

Sendo  $M$  a matriz do filtro.

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sendo  $I(x,y)$  a imagem submetida ao processamento e  $NI(x,y)$  o pixel  $(x,y)$  da imagem resultante da nova imagem é calculado da seguinte maneira:

$$NI(x,y) = I(x-1,y-1) * 1 + I(x,y-1) * 1 + I(x+1,y-1) * 1 + I(x-1,y) * 1 + \\ I(x,y) * -8 + I(x+1,y-1) * 1 + I(x-1,y+1) * 1 + I(x,y+1) * 1 + \\ I(x+1,y+1) * 1 \quad (5.4)$$

O pixel na posição horizontal imediatamente a seguir é calculado da seguinte forma:

$$NI(x+1,y) = I(x,y-1) * 1 + I(x+1,y-1) * 1 + I(x+2,y-1) * 1 + I(x,y) * 1 + \\ I(x+1,y) * -8 + I(x+2,y-1) * 1 + I(x,y+1) * 1 + I(x+1,y+1) * 1 + \\ I(x+2,y+1) * 1 \quad (5.5)$$

De reparar que de um cálculo para outro existem valores comuns, estes podem ser somados e utilizados no cálculo seguinte. Criou-se uma nova variante do pré-processamento em que a cada iteração do ciclo que percorre a imagem são calculados dois novos pixéis. Existe uma variável auxiliar ( $Va$ ) que acumula os valores comuns aos dois pixéis que depois é somada ao segundo pixel.

$$Va = I(x,y-1) * 1 + I(x+1,y-1) * 1 + I(x,y+1) * 1 + I(x+1,y+1) * 1 \quad (5.6)$$

$$NI(x,y) = I(x-1,y-1) * 1 * 1 + I(x-1,y) * 1 + I(x,y) * -8 + I(x+1,y-1) * 1 + \\ I(x-1,y+1) * 1 + Va \quad (5.7)$$

$$NI(x+1,y) = I(x+2,y-1) * 1 + I(x,y) * 1 + I(x+1,y) * -8 + I(x+2,y-1) * 1 + \\ I(x+2,y+1) * 1 + Va \quad (5.8)$$

Depois da implementação desta nova técnica conseguiu-se obter um tempo de processamento de 71 ms para imagens com 320x240 de resolução, menos 17 ms que a versão inicial o que é significativo em aplicações de tempo real.

Decidiu-se também incluir o processo de binarização a quando da convolução. Assim quando se calcula o valor do pixel da imagem resultante do processamento este é comparado com um limiar e é então efetuada a binarização. Com isto poupa-se um varrimento da imagem. Quando é detetado um pixel de valor superior ao limiar este toma o valor de luminância máxima, o que quer dizer que é um pixel de informação e consecutivamente é um pixel pertencente a uma orla.

2. Cria-se um conjunto ( $D$ ) com todos os pixéis de informação da imagem binarizada. No entanto quando se inicia este passo pela primeira vez este conjunto ( $D$ ) já está construído porque aquando da binarização da imagem, quando é definido um pixel com

luminância diferente do fundo, são já guardadas as coordenadas deste ponto. Com isto consegue-se poupar um varrimento da imagem. Depois de este passo ser executado pela primeira vez, todas as outras execuções processam-se normalmente.

3. São selecionados aleatoriamente dois pontos do conjunto D. A função utilizada é uma função que pseudoaleatória, isto porque se baseia no valor do RTC. O RTC é uma função relógio que conta o número de milissegundos decorridos após o último reset. Como são calculados dois valores aleatórios a partir do RTC em duas instruções seguidas poderá haver uma relação entre os dois valores aleatórios uma vez que a diferença de tempo entre os dois cálculos será sempre igual. No entanto em termos práticos não foram observadas limitações resultantes deste modo de cálculo de números aleatórios.
4. Verifica-se se os dois pontos estão a uma distância mínima e máxima. Caso não estejam há que regressar ao passo 2. Para diferentes resoluções utilizam-se diferentes valores de distância mínima e máxima. Para calcular a distância (D) dos dois pixéis normalmente utiliza-se a seguinte fórmula:

$$D = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} \quad (5.9)$$

O cálculo da distância foi alterado porque o cálculo da raiz quadrada é muito pesado computacionalmente. Assim não é calculada a raiz quadrada e compara-se o valor resultante com o quadrado da distância admissível.

5. Calcula-se os parâmetros  $\theta$  e  $r$  com os dois pontos anteriormente selecionados, como não existem representação de números flutuantes nas bibliotecas do *firmware* foi necessário efetuar cálculos com números inteiros. As funções cosseno e seno são calculadas a partir de uma tabela com 80 valores, o que implica uma precisão não muito alta. Contudo verificou-se experimentalmente que a precisão é suficiente para esta aplicação. As funções seno e cosseno retornam os valores multiplicados por mil.
6. Ir ao acumulador e incrementar o contador da posição ( $\theta$ ,  $r$ ) anteriormente calculados. Como por razões explicadas anteriormente não é possível trabalhar com números decimais, os valores das resoluções de  $\theta$  e  $r$  são de  $1^\circ$  e 1 pixel respetivamente. O acumulador é uma posição definida na memória de tamanho constante. Para diferentes resoluções de imagem o valor de  $r$  máximo pode ter diferentes valores dados pela seguinte equação:

$$r = \sqrt{\text{largura}^2 + \text{altura}^2} \quad (5.10)$$

Em que largura representa o número de pixéis na horizontal e altura o número de pixéis na vertical.

Já a gama de variação de  $\theta$  é sempre fixa uma vez que o ângulo da reta normal à reta a ser detetada é sempre dentro dos valores:

$$\theta \in [-180^\circ, 180^\circ]$$

Quando se incrementa o valor do contador imediatamente a seguir é verificado se o valor do contador é superior a um limiar para se poder considerar que os parâmetros descrevem uma reta existente. Assim deteta-se as retas o mais rápido possível e pode-se utilizar o número de retas detetadas como critério de paragem do algoritmo se for conhecida quantidade de retas que se pretende detetar.

Guarda-se os parâmetros da reta no vetor de retas detetadas.

Quando detetada uma reta iguala-se o valor do contador nessa posição da tabela a zero e é chamada uma rotina que irá apagar essa mesma reta da imagem. Daqui resultará uma melhor eficiência do algoritmo, uma vez que os pontos da reta detetada não irão entrar nos seguintes processos de votação. A eliminação da reta na imagem consiste em percorrer cada pixel da imagem e verificar-se a seguinte igualdade.

$$r = x * \cos(\theta) + y * \sin(\theta) \quad (5.11)$$

São utilizados os valores  $\theta$  e  $r$  da reta detetada.

Contudo com esta fórmula os tempos de processamento eram demasiados elevados (na ordem dos segundos). Visto que estes tempos de processamento impediam o uso desta *smart camera* nesta aplicação foram investigadas formas de melhorar o desempenho. Verificou-se que as funções cosseno e seno são muito “pesadas” computacionalmente.

Note-se que  $\theta$  é um valor constante para todo o varrimento da imagem. Desta forma calcula-se os valores das funções cosseno e seno com o argumento  $\theta$  antes de entrar o ciclo de varrimento. Assim a cada pixel que se efetua o teste de pertencer à reta não é calculado o valor do cosseno e seno de  $\theta$  mas são utilizado os valores já calculados antes de entrar no ciclo.

7. Verifica-se se o número de retas detetadas é igual a 3 (número de retas dos marcadores) ou se já se atingiu um número máximo de iterações para evitar ciclos, se sim o algoritmo termina aqui, senão retorna-se ao passo 1 e repete-se todo o ciclo. Como output temos o vetor de retas detetadas. Após cada execução deste processamento é necessário apagar todos os valores dos contadores do acumulador para que estes não tenham influência na votação da próxima imagem.

Como *outputs* deste processo tem-se os parâmetros da retas detetadas. Estes serão depois utilizados no pós-processamento.

### 5.2.1 EDlines

Após a implementação da transformada de Hough os tempos de processamento não foram completamente satisfatórios. Estes serão apresentados no capítulo 6 da presente dissertação. Na pesquisa por uma nova técnica de extração de linhas foi encontrado este recente algoritmo (criado em 2011) que se mostrou promissor através da comparação de tempos de processamento com outros algoritmos já existentes na literatura.

Na subsecção 2.5.2.2 foi já efetuada uma descrição pormenorizada acerca do funcionamento deste algoritmo. Nesta subsecção pretende-se descrever a implementação feita deste algoritmo. Foram feitas algumas alterações com o intuito de adaptar o algoritmo a esta aplicação e consequentemente reduzir os tempos de processamento.

Como já explicado anteriormente este algoritmo tem duas etapas principais:

1. Aplicação do algoritmo *Edge Drawing*.
2. Parametrização dos segmentos de reta resultantes da etapa anterior através do método dos mínimos quadrados.

Começa-se por descrever a implementação do *Edge Drawing* e das alterações efetuadas à versão original.

### 5.2.1.1 Edge Drawing

#### ➤ Primeiro Passo

Na versão original é feita uma convolução da imagem com um filtro Gaussiano para haver uma eliminação do ruído. Considera-se que aquisição de imagem está bem condicionada e não é necessário realizar este passo neste caso.

#### ➤ Segundo Passo

Normalmente é calculada a magnitude e direção do gradiente, sendo que para tal utiliza-se qualquer operador Sobel, Prewitt ou outros similares. O autor do algoritmo sugere a utilização de dois filtros (2x2) (um para a derivada parcial em x e outro para a derivada parcial em y). Os filtros sugeridos são os seguintes:

$$g_x(x, y) = \frac{I(x+1, y) - I(x, y) + I(x+1, y+1) - I(x, y+1)}{2} \quad (5.12)$$

$$g_y(x, y) = \frac{I(x, y+1) - I(x, y) + I(x+1, y+1) - I(x+1, y)}{2} \quad (5.13)$$

$$g(x, y) = \sqrt{g_x(x, y)^2 + g_y(x, y)^2} \quad (5.14)$$

Na versão original, como *outputs* deste passo temos duas matrizes: o mapa de gradiente que não é mais do que a magnitude do gradiente e o mapa de direção que diz que o pixel pertence a uma aresta vertical ou horizontal como explicado no capítulo de estado da arte.

No entanto nesta implementação só existe um output deste passo que é o mapa de gradiente. Explica-se agora as razões desta mudança.

Apesar do operador sugerido pelo autor deste algoritmo consistir em dois filtros de 2x2 o que é mais eficiente comparativamente aos filtros 3x3, o valor do gradiente resulta da raiz quadrada da soma das componentes ao quadrado (equação 5.14), operação essa que tornou-se demasiado pesada quando o algoritmo era executado.

Sempre com a intenção de ter o tempo de processamento o mais veloz possível não descurando a qualidade dos resultados do algoritmo, criou-se uma nova alternativa para contornar este problema. Utilizou-se o operador Canny para o cálculo do mapa de gradiente. O mapa de direção é calculado no terceiro passo. Para além da convolução com o filtro de Canny em cada pixel é efetuada uma binarização. Todos os valores abaixo de um limiar definido são igualados a 0, aos valores superiores a esse limiar nenhuma ação é tomada em relação a eles, ficando exatamente como estão. Para reduzir o tempo da convolução foi utilizada a mesma técnica apresentada na parte de pré-processamento da transformada de *Hough* aleatória desenvolvida no âmbito desta dissertação.

### ➤ Terceiro passo

Neste passo seriam calculados os *anchors* utilizando como *input* o mapa de direção e o mapa de gradiente. Contudo nesta nova variante criada existe apenas um *input*: o mapa de gradiente. O mapa de direção é calculado neste mesmo passo. O mapa de direção é utilizado no quarto passo. Apresenta-se agora em forma de código a criação do vetor de *anchors* e a criação do mapa de direção. Este código é executado para todos os pixels da imagem.

```
if (G[x, y] - G[x, y-1] >= ANCHOR_THRESH && G[x, y] - G[x, y+1] >= ANCHOR_THRESH) {
    anchor = true; // se respeitar os limites é um anchor
    D[x, y] == HORIZONTAL;
}

if (G[x, y] - G[x-1, y] >= ANCHOR_THRESH && G[x, y] - G[x+1, y] >= ANCHOR_THRESH) {
    anchor = true; // se respeitar os limites é um anchor vertical
    D[x, y] == VERTICAL;
}

Else{
    anchor = false; // não se verifica, não é anchor
}
```

O limiar ANCHOR\_THRESH é por norma igual a 8. Mais uma vez dependendo do tipo de aplicações e do género de imagens este valor pode ser alterado. No caso desta aplicação verificou-se na prática que este limite era o aconselhado.

A principal diferença entre esta nova técnica desenvolvida e a inicial é que primeiramente era testado se o pixel era horizontal ou vertical e depois verificavam-se as condições para ver se era um *anchor*. Neste nova técnica primeiramente testa-se as condições e se estas forem verdadeiras define-se a direção do *anchor*. A grande vantagem desta técnica é que o mapa de direção não é calculado para todos os pixels, mas apenas para os *anchors* que são pixels que têm alta probabilidade de pertencer a um edge. Com isto é novamente reduzido o tempo de processamento de todo o algoritmo.

### ➤ Quarto passo

Depois de estar definido o vetor de *anchors*, a etapa seguinte é fazer a ligação dos *anchors*. É neste momento que se desenham as várias retas e se guarda os pixéis pertencentes à reta num vetor. Para o fazer são verificados os valores dos 3 pixéis adjacentes e escolhido o pixel com maior valor em termos de luminância.

Para cada *anchor*, dependendo se este é um *anchor* pertencente a um *edge* vertical ou horizontal é efetuado o processo de “desenho” da reta através do varrimento horizontal ou vertical. As retas existentes no *landmarks* a serem detetados tocam mas não chegam a interseção o que faz com que não seja necessário testar a direção de todos os pixéis.

Para além da modificação anteriormente referenciada foi ainda adicionada uma nova variável ao algoritmo (*NumeroPixeis*) que é incrementada à medida que novos pixéis vão sendo adicionados ao segmento de reta que esta a ser “desenhado”. Esta variável é depois utilizada no pós-processamento. Para além disso, se o tamanho mínimo das retas que se pretende detetar for conhecido pode-se definir um número mínimo de pixéis necessários para se considerar uma reta detetada. Isto pode fazer com que se evitem falsas deteções.

De seguida apresenta-se o processo de “desenho” dos diferentes segmentos de reta a partir de *anchors*. Apresenta-se apenas o procedimento de “desenho” de uma reta horizontal para a esquerda, no entanto, o desenho das retas nas outras direções é análogo ao aqui apresentado. Depois de se completar o “desenho” da reta para a esquerda volta-se ao *anchor* que deu início ao processo e continua-se a desenhar segmento de reta para a direita enquanto as condições forem verificadas.

```
while (G[x, y] > 0 && E[x, y] != EDGE){
    E[x, y] = EDGE;          // marcar o pixel como um pixel pertencente a uma reta e
                             // guarda-lo num vetor
    NumeroPixeis++; // incrementa o valor de pixéis existentes na reta
    // verifica os valores dos três pixéis à esquerda
    if (G[x-1, y-1] > G[x-1, y] && G[x-1, y-1] > G[x-1, y+1]){
        x = x-1; y = y-1;    // é escolhido o pixel superior esquerdo
    } else if (G[x-1, y+1] > G[x-1, y] && G[x-1, y+1] > G[x-1, y-1]){
        x = x-1; y = y+1;    // é escolhido o pixel inferior esquerdo
    } else {
        x = x-1;              // é escolhido o pixel à esquerda
    }
}
```

## ❖ Segunda etapa

Originalmente nesta etapa a partir dos vetores anteriormente criados e com as coordenadas existentes nos mesmos era efetuado o cálculo de uma parametrização para as retas através do método dos mínimos quadrados.

A rotina do método dos mínimos quadrados veio revelar-se muito pesada e uma vez mais tentou-se encontrar uma nova forma de contornar este problema. Atendendo ao facto que as retas a serem detetadas não apresentam uma dispersão de pontos muito alta testou-se o cálculo dos parâmetros  $\theta$  e  $r$  a partir dos dois pontos mais distantes (entre si) existentes na reta, isto tornou-se possível devido ao conhecimento do número de pixéis existentes em cada reta, funcionalidade implementada no quarto passo da primeira etapa. Na prática esta nova técnica revelou resultados satisfatórios e substituiu-se o método dos mínimos quadrados.

Após este processamento de imagem tem-se os parâmetros das retas detetadas e o número de pixéis que constituem cada uma.  $\theta$

### 5.2.3 Pós-processamento

Depois da extração dos parâmetros das retas obtidos tanto através da transformada de *Hough* como a partir do *EDlines* é necessário passar por um processo de validação das mesmas e tratamento de dados.

Neste processamento pretende-se verificar a validade das retas detetadas, que tipo de tipo de *landmark* foi detetado (triângulo ou linhas), a posição relativa e o desvio da *landmark* em relação ao robô. De acordo com a tabela 4.1 era também objetivo que a localização absoluta fosse um *output* deste processamento. Contudo não foi possível integrar a câmara no robô por escassez de tempo, o que impediu o estabelecimento de uma relação entre a distância e o número de pixéis das várias posições dos marcadores de bits existentes dentro do marcador. Para teste dos processamentos implementados foram utilizadas imagens captadas através dos sistemas desenvolvidos pelos estudantes David Lima [38] e Roberto Silva [4]. Os resultados encontram-se disponíveis no capítulo 6 desta dissertação. Este algoritmo de pós-processamento é baseado nos processamentos anteriormente desenvolvidos por aqueles estudantes.

Primeiramente verifica-se se as retas detetadas formam um *landmark* do tipo linha. Esta verificação é feita através da comparação dos ângulos das retas detetadas. Se existirem duas retas com ângulos similares dentro de um limite de tolerância de erro, então existem duas retas paralelas e considera-se que foi detetada uma linha. Caso contrário efetuam-se mais testes para verificar se as retas detetadas formam um marcador do tipo triângulo.

Considerando que as foram detetadas duas linhas com ângulos muito similares são efetuados os seguintes procedimentos:

- 1- Calcula-se o ângulo desvio da linha. Este cálculo é muito simples de se efetuar visto que corresponde ao ângulo que parametriza as retas mais longas que definem a linha. Assim o ângulo de desvio é calculado através da média dos ângulos das duas retas.

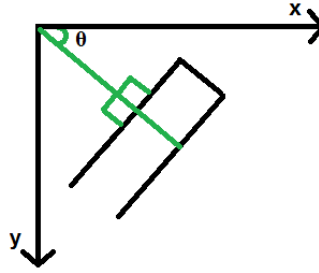


Figura 5.3 - Desvio em retas paralelas.

- 2- Depois de já se ter calculado o desvio angular falta apenas calcular a posição relativa da linha detetada [38]. Como uma linha é demasiado comprida para o campo de alcance da câmara, uma imagem nunca irá conter a totalidade do *landmark*. Isto levou a que se simplificasse o processo de determinação do ponto central (posição relativa) da *landmark*. Sendo assim quando as retas apresentam um ângulo de desvio no intervalo  $[-45^\circ, 45^\circ]$  o ponto central da linha é dado por  $(x_c, y_c)$ , com  $y_c = (\text{imgHeight}/2)$  e  $x_c$  dado pela equação 5.14. Se o desvio angular encontrar-se fora do intervalo anteriormente referido o ponto central da linha é dado por  $x_c = 0$  e  $y_c$  dado pela fórmula 5.15.

$$x_c = -\tan(\theta) * y_c + \frac{\rho_r}{\cos(\theta)} \quad (5.15)$$

$$\rho_r = \frac{\max(\rho_{reta1}, \rho_{reta2}) - \min(\rho_{reta1}, \rho_{reta2})}{2} + \min(\rho_{reta1}, \rho_{reta2}) \quad (5.16)$$

Desta forma se conclui o pós-processamento deste tipo de *landmarks*. Passa-se agora ao pós-processamento dos *landmarks* do tipo triângulo.

Este processo inicia-se quando não são detetadas duas retas paralelas nas retas que são fornecidas pela transformada de *Hough* aleatória ou pelo *EDlines*. Numa primeira fase são efetuados testes para verificar se as retas detetadas realmente definem o marcador procurado. O marcador tem forma de triângulo. Um teste fácil para verificar a validade das retas é aferir se a soma dos ângulos que as retas fazem entre elas é igual a  $180^\circ$  (mais ou menos uma determinada tolerância), uma vez que a soma dos ângulos internos de um triângulo é sempre igual a  $180^\circ$ . Pelo teorema do ângulo externo [44], é possível calcular os ângulos de cada vértice através da seguinte fórmula:

$$\alpha_{vert\_reta1,reta2} = \arctan \left( \left| \frac{\tan(\theta_{reta1}) - \tan(\theta_{reta2})}{1 + \tan(\theta_{reta1}) * \tan(\theta_{reta2})} \right| \right) \quad (5.17)$$

Se os dados das retas passarem pelo teste anterior está provado que estamos perante um triângulo. No entanto existe a hipótese de as retas formarem um triângulo que não corresponde ao *landmark* a ser detetado. Para isso é feito um teste ao comprimento das arestas do triângulo para verificar se estas possuem o tamanho dos *landmarks* a serem detetados através das seguintes fórmulas [4]:

$$x_{vertice\ reta1,reta2} = \frac{\rho_2 \sin(\theta_1) - \rho_1 \sin(\theta_2)}{\cos(\theta_2) \sin(\theta_1) - \cos(\theta_1) \sin(\theta_2)} \quad (5.18)$$

$$y_{vertice\ reta1,reta2} = \frac{\rho_1}{\sin(\theta_1)} - \frac{\cos(\theta_1)}{\sin(\theta_1)} \times x_{vert\ reta1,reta2} = \frac{\rho_2}{\sin(\theta_2)} - \frac{\cos(\theta_2)}{\sin(\theta_2)} \times x_{vert\ reta1,reta2} \quad (5.19)$$

$$Tamanho = \sqrt{(y_2 + y_1)^2 + (x_2 + x_1)^2} \quad (5.20)$$

O tamanho é então comparado com o tamanho esperado e se o tamanho da reta detetada estiver dentro de um intervalo aceitável considera-se que as retas detetadas definem o marcador do tipo triângulo. Com isto conclui-se o processo de validação das retas. Passa-se agora ao cálculo do desvio angular do marcador. Este desvio angular é dado pelo ponto médio da aresta mais pequena do marcador e pelo vértice oposto a esta mesma aresta. Primeiramente calcula-se o ponto médio da reta mais pequena:

$$x_{pint} = \left( \frac{|x_1 - x_2|}{2} \right) + \min(x_1 + x_2) \quad (5.21)$$

$$y_{pint} = \left( \frac{|y_1 - y_2|}{2} \right) + \min(y_1 + y_2) \quad (5.22)$$

Em que  $x_{pint}$  e  $y_{pint}$  são as coordenadas a serem calculadas do ponto intermédio da aresta mais pequena do triângulo e  $y_1, y_2, x_1$  e  $x_2$  são as coordenadas dos dois vértices da mesma aresta, já calculadas anteriormente.

Para o cálculo do desvio do marcador são utilizadas as coordenadas do ponto médio da aresta mais pequena calculadas no passo imediatamente anterior e o terceiro vértice que não foi utilizado no passo anterior. Este terceiro vértice é o que une as outras duas retas do marcador como ilustra a próxima figura.

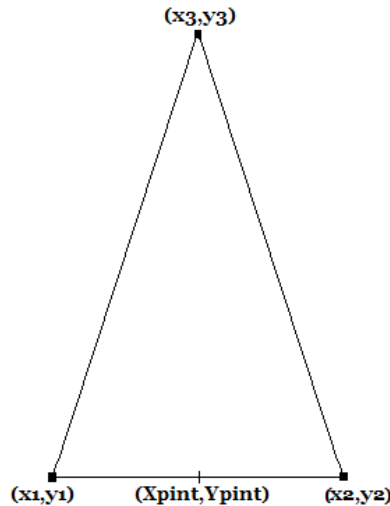


Figura 5.4 - Representação dos vértices dos marcadores.

O ângulo do segmento de reta que une o ponto intermédio com o vértice 3 define então o desvio do marcador. O ângulo é dado pela seguinte fórmula:

$$\theta_{desvio} = \text{atan} \left( \frac{y_3 - y_{int}}{x_3 - x_{int}} \right) \quad (5.23)$$

A partir deste momento falta apenas calcular a localização relativa que é o ponto central do marcador e a localização absoluta. Como explicado no início desta subsecção não foi possível implementar a descodificação da localização absoluta porque não houve tempo suficiente para integrar a *smart camera* no robô e faltou fazer a relação entre a distância e o número de pixels que representam essa distância na imagem para as diferentes resoluções em que a câmara opera. A localização relativa é calculada pelas seguintes fórmulas:

$$x_{Pcentral} = |x_{P3} - x_{Pint}| + \min(x_{P3}, x_{Pint}) \quad (5.24)$$

$$y_{Pcentral} = |y_{P3} - y_{Pint}| + \min(y_{P3}, y_{Pint}) \quad (5.25)$$

Em que  $y_{Pcentral}$  e  $x_{Pcentral}$  são as coordenadas do ponto central do marcador e consecutivamente as coordenadas da localização relativa,  $y_{P3}, x_{P3}$  são as coordenadas do vértice 3 como exemplifica a figura 5.4 e  $y_{Pint}, x_{Pint}$  são as coordenadas do ponto intermédio da aresta mais pequena do triângulo.

Com isto conclui-se o algoritmo de pós-processamento implementado.

Apresenta-se agora um esquema que visa fornecer um resumo funcional de todo o processamento implementado.

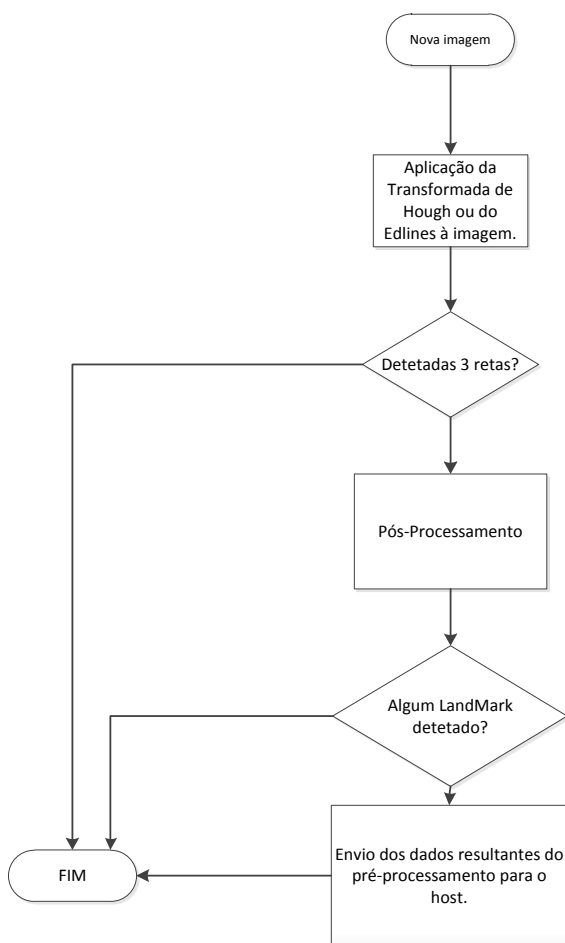


Figura 5.5 - Diagrama funcional dos processamentos implementados.

Assim conclui-se esta secção de implementação dos algoritmos de processamento de imagem. Todos os resultados relativos a estas implementações encontram-se no capítulo seguinte.

# Capítulo 6

## Resultados

Neste capítulo pretende-se apresentar os resultados obtidos com as implementações dos algoritmos apresentados no capítulo 5. Inicialmente serão apresentados os impactos que as melhorias sucessivas efetuadas tiveram em termos de tempos de processamento, depois serão apresentados os tempos de processamento das várias etapas de cada algoritmo para diferentes resoluções. Seguidamente, far-se-á uma análise comparativa entre os tempos de cada processamento. Por fim apresenta-se uma comparação à robustez do algoritmo EDlines e da transformada de Hough através do processamento de imagens com ruído e de marcadores com desgaste.

### 6.1 Impacto das melhorias implementadas

Nesta subsecção pretende-se quantificar as melhorias implementadas. Para além da implementação dos processamentos houve um grande esforço para se reduzir os tempos de processamento continuamente. Estas melhorias foram realizadas utilizando imagens com 320x240 de resolução e os resultados apresentados nesta secção são relativos a esta resolução, mais tarde serão apresentados os resultados dos algoritmos finais para as três resoluções suportadas pela *smart camera*.

#### 6.1.1 Impactos na transformada de Hough

Como descrito na subsecção 5.2.1.1 foi feito uma melhoria na implementação do filtro de *Canny*. Antes da implementação esta rotina demorava cerca de 89 milissegundos que é um tempo muito significativo tendo em conta a duração total do processamento.

Após as melhorias efetuadas esta rotina passou a ter a duração de 71 milissegundos. Trata-se de uma poupança de aproximadamente 20% o que é significativo em processamentos que se pretendam que trabalhem em tempo real.

Note-se que a rotina atrás referida além de efetuar a convolução da imagem com um filtro *Canny* também efetua a binarização da imagem e guarda os pixéis com a luminância

máxima (pontos de informação) num vetor. As três operações são efetuadas em apenas um varrimento da imagem poupando-se assim dois varrimentos que iriam ser desnecessários, uma vez que estas operações podem ser efetuadas conjuntamente.

Uma outra melhoria implementada foi na rotina que elimina a reta da imagem depois de esta ser detetada, como explicado no passo 6 da subsecção 5.2.1.1.

Inicialmente a fórmula de teste para verificar se um determinado pixel pertencia à reta utilizava funções cosseno e seno que tinham como argumento o ângulo da reta detetada como mostra a equação 5.11. Em uma imagem com resolução 320x240 as funções cosseno e seno eram calculadas 76800 vezes. Sendo que estas funções são algo pesadas relativamente às outras utilizadas no algoritmo havia aqui muito potencial para reduzir tempos de processamento. Assim calcula-se o valor dos cossenos e do seno do ângulo de entrar no varrimento e utiliza-se esse resultado para o teste efetuado a cada pixel. Assim já não se calcula os cosseno e seno para cada pixel mas utiliza-se o resultado dos mesmos calculados previamente. Antes da implementação desta melhoria o processo de eliminação da reta demorava em média 350 milissegundos o que numa imagem em que existem 3 retas a ser detetadas só o processo de eliminação das retas demoraria mais de um segundo. Após a implementação do novo processo o processo de eliminação de uma reta na imagem tem a duração constante de 3 milissegundos. Na próxima tabela apresentam-se o resumo destas melhorias na transformada de Hough.

**Tabela 6.1 - Tabela resumo das melhorias implementadas na transformada de Hough.**

|                           | Antes   | Depois | Melhoria |
|---------------------------|---------|--------|----------|
| <b>Convolução</b>         | ≈89 ms  | ≈71 ms | ≈20,2%   |
| <b>Eliminação da reta</b> | ≈350 ms | ≈3 ms  | ≈99,1%   |

### 6.1.2 Impactos no algoritmo EDlines

O algoritmo EDlines foi também sujeito a sucessivas melhorias. Inicialmente havia o problema da escolha do filtro a implementar. Como descrito no passo 2 da subsecção 5.2.1.1 o autor [31] do processamento sugere a utilização do filtro das equações 5.12, 5.13 e 5.14. Após a sua implementação verificou-se que este tinha um tempo de processamento muito alto. O problema não estava no filtro em si mas na raiz quadrada da soma das componentes ao quadrado. Este processo era muito pesado e a implementação deste filtro demorava cerca de 300 milissegundos o que inviabilizava a utilização deste algoritmo para este processamento. Assim efetuou-se a substituição do filtro sugerido pelo operador Canny. Esta substituição teve implicações no restante algoritmo como explicado na subsecção 5.2.1.1 visto que do operador sugerido pelo autor resultavam as componentes das derivadas parciais ao longo dos dois eixos da imagem e no operador implementado é obtido apenas o mapa de magnitude da imagem.

A implementação do filtro *Canny* foi feita da mesma forma que o pré-processamento da transformada de *Hough*. A única diferença é que neste caso não é necessário guardar as

coordenadas dos pixels de informação. Com esta implementação deste filtro obteve-se um tempo de processamento de 69 milissegundos.

Outra alteração feita ao algoritmo proposto pelo autor reside na substituição do método dos mínimos quadrados para parametrizar os segmentos de reta detetados. Foi verificado que a aplicação do método dos mínimos quadrados levava um tempo considerável a processar (cerca de 200 milissegundos para 3 retas). Atendendo às características das retas que se pretende detetar decidiu-se abdicar do método dos mínimos quadrados e utilizar o primeiro e último ponto da reta detetada para calcular a equação da reta. Este novo procedimento tem um tempo de processamento de 3 milissegundos para 3 retas. Na prática verificou-se que este método era satisfatório como ilustra a seguinte figura.

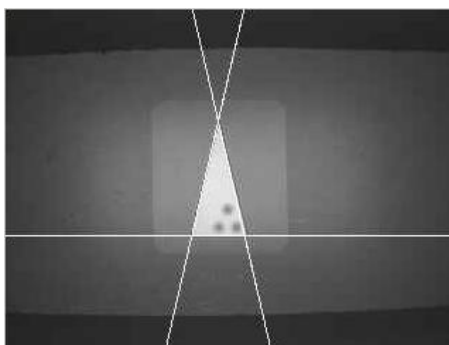


Figura 6.1 - Resultado da parametrização com o uso de apenas 2 pontos.

Apresenta-se agora uma tabela com as melhorias efetuadas.

Tabela 6.2 - Tabela resumo das melhorias implementadas no EDlines.

|                               | Antes    | Depois  | Melhoria |
|-------------------------------|----------|---------|----------|
| <b>Mapa Magnitude</b>         | ≈ 300 ms | ≈ 69 ms | ≈ 77,0%  |
| <b>Parametrização da reta</b> | ≈ 200 ms | ≈ 3 ms  | ≈ 98,5%  |

## 6.2 Tempos de processamento dos algoritmos e suas etapas

Nesta secção pretende-se apresentar os resultados dos tempos dos processamentos implementados e das suas várias etapas para as três resoluções de imagem suportadas pela câmara: 160x120, 320x240 e 640x480. Os resultados (tempos de processamento) serão separados por algoritmo, e no fim de cada apresentação será feita uma análise desses mesmos tempos. No final da exibição dos resultados será feita uma análise comparativa entre os tempos de processamento dos dois algoritmos.

### 6.2.1 Tempos de processamento da Transformada de *Hough* Aleatória

Para a obtenção destes valores foi feita a média dos valores obtidos em 10 imagens com diferentes orientações dos *landmarks*.

Tabela 6. 3 - Tempos de processamento nas várias resoluções.

| Etapas / Resolução                  | 160x120              |                                          | 320x240              |                                          | 640x480              |                                          |
|-------------------------------------|----------------------|------------------------------------------|----------------------|------------------------------------------|----------------------|------------------------------------------|
|                                     | <u>Tempo</u><br>(ms) | $\frac{\text{Tempo}}{\text{Temp.Total}}$ | <u>Tempo</u><br>(ms) | $\frac{\text{Tempo}}{\text{Temp.Total}}$ | <u>Tempo</u><br>(ms) | $\frac{\text{Tempo}}{\text{Temp.Total}}$ |
| Pré-processamento                   | 15                   | 21,7%                                    | 71                   | 41,0%                                    | 295                  | 52,7%                                    |
| Eliminação das três retas da imagem | 3                    | 4,3%                                     | 9                    | 5,3%                                     | 33                   | 5,9%                                     |
| Procura pixéis de informação        | 6                    | 8,7%                                     | 12                   | 6,9%                                     | 75                   | 13,4%                                    |
| Reset dos valores do acumulador     | 1                    | 1,5%                                     | 1                    | 0,6%                                     | 2                    | 0,4%                                     |
| Processo de Votação                 | 36                   | 52,2%                                    | 72                   | 41,6%                                    | 147                  | 26,2%                                    |
| Pós-processamento                   | 8                    | 11,6%                                    | 8                    | 4,6%                                     | 8                    | 1,4%                                     |
| Total                               | 69                   | 100%                                     | 173                  | 100%                                     | 560                  | 100%                                     |

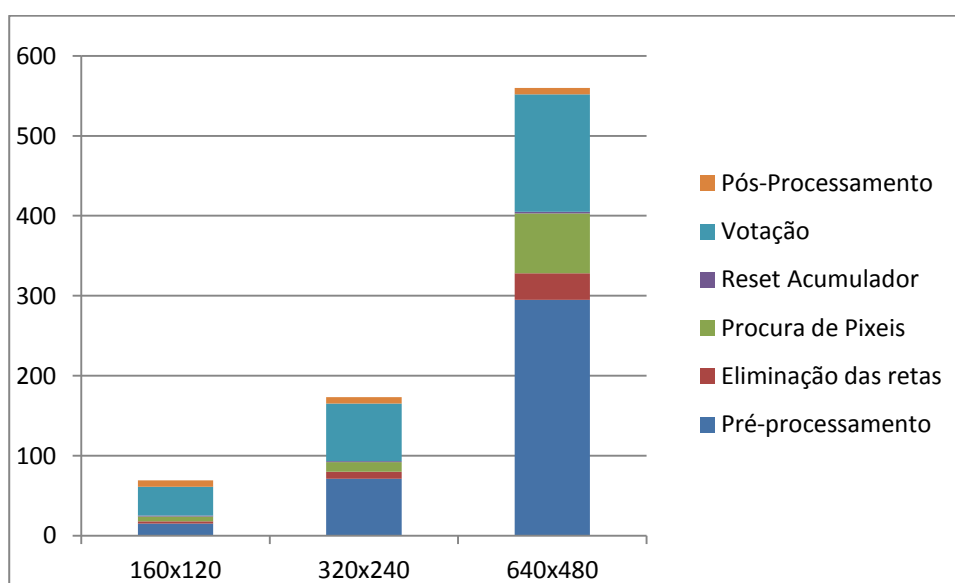


Figura 6. 2 - Distribuição dos tempos de processamento das várias etapas. Tempo em ms.

Como seria de esperar os tempos crescem com o aumento das resoluções das imagens. Passa-se agora a uma análise mais detalhada da variação dos tempos das diferentes etapas da transformada de Hough.

Tanto o tempo de pré-processamento como o tempo de votação crescem com a resolução porque existem mais pixéis a processar. Repare-se, no entanto, que o pré-processamento vai aumentando em termos percentuais em relação ao tempo total enquanto a percentagem do tempo de votação vai diminuindo. Isto acontece porque com o aumento das resoluções existem mais pixéis para efetuar a convolução do filtro. No caso do tempo de votação, os pixéis de informação não aumentam da mesma maneira que a resolução, então a percentagem de tempo que esta tarefa ocupa vai diminuindo com o aumento da resolução e diminuição dos pixéis de informação em termos percentuais.

Repare-se que o tempo de pré-processamento aumenta aproximadamente 4 vezes de uma resolução para outra. Isto acontece porque a altura e a largura das imagens aumenta 2 vezes entre duas resoluções consecutivas (ex: 160x120 e 320x240, 320x240 e 640x480) o que resulta numa área de pixéis 4 vezes superior. É então necessário fazer a convolução do filtro em 4 vezes mais pixéis e como seria de esperar isto traduz-se num tempo 4 vezes maior.

Também o tempo de procura de pixéis de informação aumenta com o incremento da resolução, mas em termos percentuais diminui devido ao mesmo fator da diminuição do tempo de votação.

O tempo de eliminação das retas torna-se maior porque existem mais pixéis a eliminar com o aumento da resolução.

O tempo do *reset* do acumulador cresce com o aumento da resolução mas não significativamente. O tamanho do acumulador é dado pela fórmula:

$$Tamanho_{acumulador} = \frac{\rho_{max}}{Resolução_{\rho}} * \frac{180}{Resolução_{\theta}} \quad (6.1)$$

O valor de  $\rho_{max}$  é dado pela equação 5.10 e os valores de  $Resolução_{\rho}$  e  $Resolução_{\theta}$  são os dois iguais a 1.

Como o tempo de processamento não aumenta significativamente, conclui-se que é um processamento muito rápido e que não terá influência na utilização deste mesmo algoritmo em resoluções superiores às apresentadas.

Por fim, verifica-se que o tempo de pós-processamento é constante. Tal acontece porque como input deste processo temos os parâmetros das retas detetadas. Oras as retas detetadas são sempre 3 ou menos. Desta forma os cálculos efetuados no pós-processamento têm sempre o mesmo volume.

## 6.2.2 Tempos de processamento do EDlines

Para a obtenção destes valores foi feita a média dos tempos obtidos em 10 imagens com diferentes orientações dos *landmarks*.

Tabela 6.4 - Tempos de processamento nas várias resoluções.

| Etapas / Resolução               | 160x120              |                                   | 320x240              |                                   | 640x480              |                                   |
|----------------------------------|----------------------|-----------------------------------|----------------------|-----------------------------------|----------------------|-----------------------------------|
|                                  | <u>Tempo</u><br>(ms) | <u>Tempo</u><br><u>Temp.Total</u> | <u>Tempo</u><br>(ms) | <u>Tempo</u><br><u>Temp.Total</u> | <u>Tempo</u><br>(ms) | <u>Tempo</u><br><u>Temp.Total</u> |
| Pré-processamento                | 14                   | 46,7%                             | 69                   | 63,9%                             | 290                  | 67,2%                             |
| Procura de Anchors               | 6                    | 20,0%                             | 29                   | 26,9%                             | 132                  | 30,3%                             |
| Desenho das retas                | 1                    | 3,3%                              | 1                    | 0,9%                              | 1                    | 0,3%                              |
| Cálculo dos parâmetros das retas | 1                    | 3,3%                              | 1                    | 0,9%                              | 1                    | 0,3%                              |
| Pós-processamento                | 8                    | 26,7%                             | 8                    | 7,4%                              | 8                    | 1,9%                              |
| Total                            | 30                   | 100%                              | 108                  | 100%                              | 432                  | 100%                              |

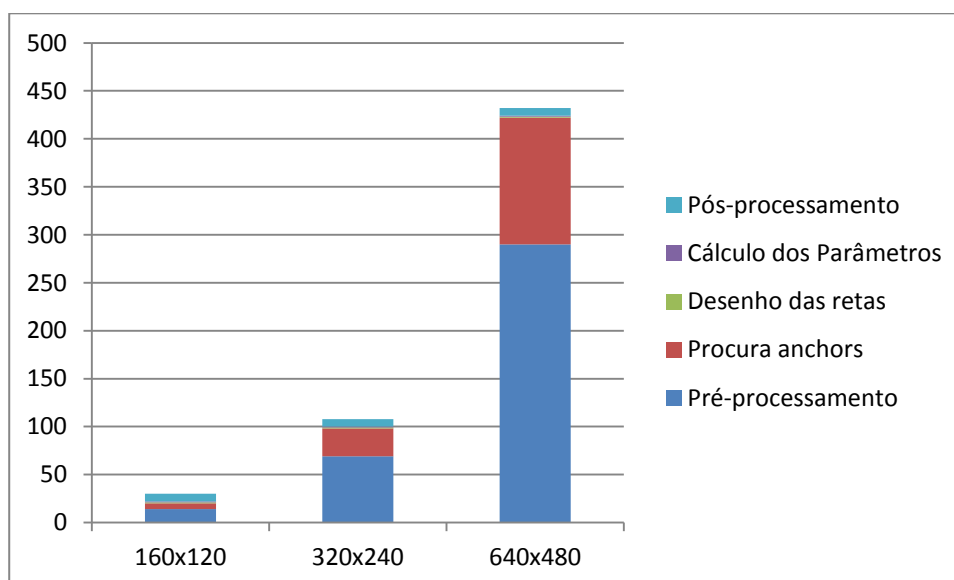


Figura 6. 3 -Distribuição dos tempos de processamento das várias etapas EDlines. Tempo em ms.

Verifica-se que o tempo de pré-processamento é o que apresenta uma maior influência no tempo de processamento. Tal como era de esperar os tempos aumentam 4 vezes mais de resolução para resolução devido ao facto de o número convoluções com cada pixel aumentar 4 vezes também.

O tempo de desenho das retas e o de cálculo dos parâmetros não são significativos pelo que pouco ou nenhum impacto têm no tempo total.

O passo de procura de *anchors* é também um dos passos que demora mais tempo a processar. Isto ocorre porque este processo percorre todos os pixéis da imagem e faz o teste para verificar se o pixel é um *anchor* ou não.

### 6.3 Comparação dos tempos de processamento

Pretende-se apresentar nesta secção uma análise comparativa em termos temporais dos dois algoritmos implementados.

**Tabela 6.5 - Resumos dos tempos de processamento totais.**

| Algoritmos/Processamento       | 160x120 | 320x240 | 640x480 |
|--------------------------------|---------|---------|---------|
| Transf. <i>Hough</i> Aleatória | 69 ms   | 173 ms  | 560 ms  |
| EDlines                        | 30 ms   | 108 ms  | 432 ms  |

No geral verifica-se que o EDlines é mais rápido que a transformada de Hough aleatória. No entanto, os tempos não são tão diferentes em resoluções mais altas. A razão disto acontecer está no facto de a etapa que demora mais tempo a ser processada é o pré-processamento, etapa essa que é comum aos dois algoritmos.

De notar também que a transformada de Hough aleatória apresenta um tempo variável consoante o número de retas a serem detetadas. Para uma imagem de resolução 320x240, o processo de eliminação de cada reta demora cerca de 3 ms e o processo de procurar pixéis de informação da imagem demora cerca de 4 ms. Cada uma destas funções é executada cada vez que é detetada uma nova reta. Para cada reta a ser detetada o algoritmo irá demorar 4 ms + 3 ms = 7 ms a mais. Contudo, não é só este tempo que aumenta. Repare-se que ao haver mais retas irão entrar mais pontos no processo de votação. Existindo mais pontos no processo de votação serão necessárias mais iterações para ser detetada uma reta. O que originará um aumento do processamento total da imagem.

Já em relação ao EDlines, o seu tempo de processamento não é tão variável. A etapa de desenho das retas e a de cálculo de parâmetros apresentam um tempo de processamento muito baixo, pelo que o aumento de retas presentes na imagem não se traduzirá em impactos temporais significativos. As etapas de procura de *anchors*, de pré-processamento e de pós-processamento têm um tempo de execução constante independente do número de retas presentes na imagem, visto que estas apenas varrem a totalidade da imagem e efetuam testes/operações em cada pixel independentemente dos valores dos mesmos.

## 6.4 Comparação da robustez dos processamentos

A comparação de tempos de processamento não é suficiente para comparar a eficácia e eficiência dos mesmos. Desta forma decidiu-se analisar a robustez dos algoritmos com a utilização de imagens com ruído e de imagens com os marcadores destruídos.

Primeiramente utilizou-se imagens com os marcadores danificados.

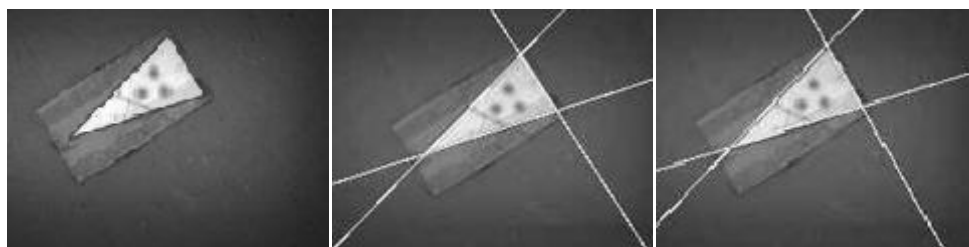


Figura 6.4 - Imagem utilizada à esquerda, processamento RHT ao centro e EDlines à direita

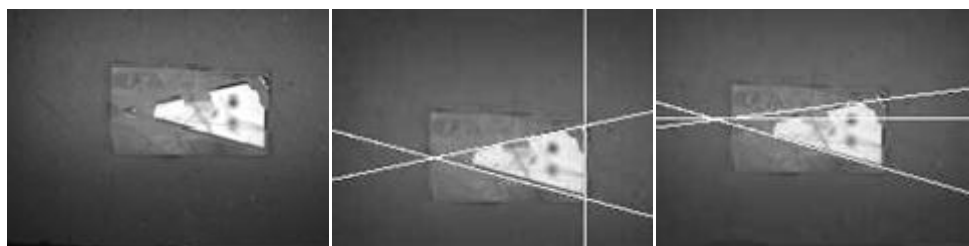


Figura 6.5 - Imagem utilizada à esquerda, processamento RHT ao centro e EDlines à direita.



Figura 6.6 - Imagem utilizada à esquerda, processamento RHT ao centro e EDlines à direita.

Nas figuras acima apresentadas pode-se verificar que a transformada de Hough aleatória apresenta melhores resultados do que o EDlines para imagens com marcadores danificados. Isto deve-se sobretudo ao facto de que para haver a detecção de uma reta no algoritmo EDlines é necessário a mesma ser contínua para o desenho desta poder ser efetuado na totalidade. Ao haver um ponto de descontinuidade, o desenho da reta pára nesse mesmo local. Já na transformada de Hough aleatória são seleccionados vários pontos aleatórios pelo que a descontinuidade das retas em alguns pontos não terá influência no resultado final.

Passa-se agora a uma análise do processamento de imagens com ruído. Para adição de ruído utilizou-se o *software Matlab*.

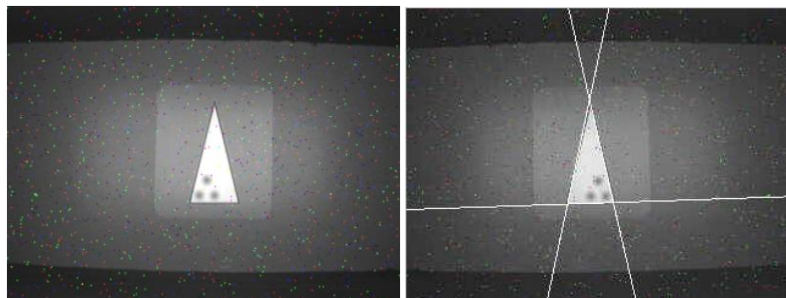


Figura 6. 7 - Imagem com densidade de ruído uniforme de 0,03. Resultado do processamento Edlines à direita.

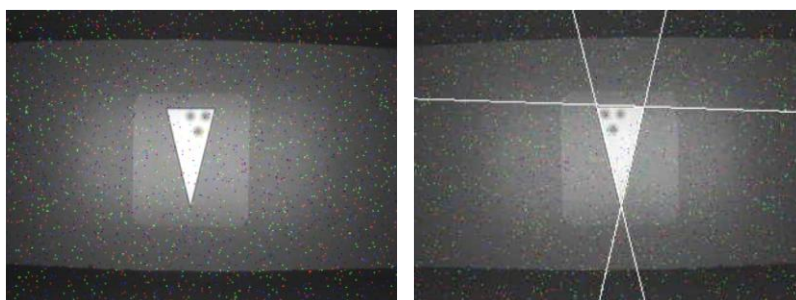


Figura 6. 8 - Imagem com densidade de ruído uniforme de 0,05. Resultado do processamento Edlines à direita.

Os resultados da transformada de Hough aleatória não foram apresentados nas figuras acima porque não se conseguiu detetar retas. A não deteção de retas ocorre porque com o ruído existem muitos pixéis que são considerados pontos de informação erradamente, ou seja, estes pixéis entram também para o processo de votação. Como estes pontos não pertencem a retas da imagem mas sim a ruído, irão aumentar significativamente o tempo de votação. Neste caso, o número de iterações máximas do algoritmo foi atingido sem que se tenha detetado alguma reta. Aumentando o número de interações máximas, as retas acabariam por ser detetadas. No entanto, o tempo de execução iria aumentar muito o que tornaria o processamento inviável.

Com o EDlines obteve-se resultados razoáveis como ilustram as figuras 6.6, 6.7 e 6.8, uma vez que o ruído tem pouca influência no algoritmo. De notar que as parametrizações das retas não são as mais desejadas. Em casos de análise de imagens com grande quantidade de ruído deve-se utilizar o método dos mínimos quadrados para parametrização das retas. Observou-se que o EDlines deixa de detetar as retas dos marcadores quando o ruído é tão intenso que provoca descontinuidades das retas a serem detetadas.



# Capítulo 7

## Conclusões

Tendo em conta tudo o que foi apresentado ao longo deste documento é possível afirmar que os resultados obtidos são satisfatórios.

Primeiramente, produziu-se três tutorias que permitem informar futuros utilizadores da *smart camera* acerca da interação com esta e da criação de um ambiente de desenvolvimento adequado para implementação de processamentos de imagem.

O estudo e descrição do *hardware* e *software* permitiram adquirir conhecimentos sobre o sistema que viriam a ser úteis na parte de construção de rotinas de análise de imagem. Tal descrição pode vir também a ser muito útil no futuro para o desenvolvimento de novas aplicações.

A aplicação desenvolvida em *Lazarus* apresenta-se como um modo de interagir com a câmara, muito útil na definição de parâmetros de processamentos, medição de tempos de execução e envio de imagens para processamento *on board*.

Devido à escassez de tempo para realizar todas as tarefas desta dissertação não foi possível integrar a *smart camera* no robô. Para além disso, foi investido muito tempo na investigação e criação de melhorias aos processamentos de imagem criados no âmbito desta dissertação. Sem as melhorias implementadas, os processamentos não poderiam ser utilizados nesta *smart camera* o que iria inviabilizar a sua posterior integração no AGV.

Considera-se que as melhorias construídas foram satisfatórias. No entanto, existe ainda potencial para melhorar todo o algoritmo. Como foi verificado, o pré-processamento ocupa uma grande percentagem do tempo total da execução dos algoritmos, pelo que será a parte em que mais melhorias podem ser realizadas.

Através da análise dos resultados obtidos pode dizer-se que em geral a transformada de *Hough* aleatória é mais lenta que o EDlines. Em termos de robustez, o EDlines apresenta melhores resultados para imagens com ruído e a transformada de *Hough* aleatória apresenta melhor *performance* em imagens com marcadores danificados.

Considera-se também que este documento constitui também uma boa comparação entre os dois processamentos de imagem apresentados nomeadamente nas implementações em *smart cameras*.

Em suma crê-se que foi criada uma documentação que permite ter um conhecimento mais aprofundado acerca da *Surveyor Blackfin Camera* e um *firmware* útil para aplicações que envolvam a deteção de retas.

## 7.1 Trabalhos futuros

Devido à duração limitada desta dissertação não foi possível integrar a câmara no robô, pelo que esta integração é considerada o passo mais importante a ser desenvolvido no futuro. Para tal, é necessário realizar vários testes aos processamentos de imagem (uma vez realizada a integração) e ainda adaptar os meios de comunicação que serão utilizados para trocar informação com a aplicação de localização e controlo.

Devido ao tempo de execução do pré-processamento considera-se também que é necessário implementar mais melhorias nesta etapa de processamento para obtenção de melhores resultados globais.

Os processamentos de imagem implementados podem ser utilizados para uma vasta área de aplicações onde haja a necessidade de deteção de retas.

Aconselha-se a aquisição de um dispositivo de interface JTAG para o desenvolvimento de futuros projetos utilizando esta *smart camera*, uma vez que iria melhorar o processo de *debugging*. Sem o dispositivo JTAG o *debugging* torna-se muito moroso.

# Anexo A

## Compilação e Transferência do Firmware em Linux

Neste tutorial serão apresentados todos os passos e programas necessários à compilação e transferência do *firmware* para a *smart camera*.

São necessários os seguintes *softwares*:

- *Blackfin Toolchain* - Contém as bibliotecas necessárias à compilação do software.
- *Moserial* - *Software* de emulação de terminal que irá permitir efetuar a comunicação via porta série e transferir o ficheiro resultante da compilação.

### 1. Instalação da Blackfin Toolchain

1. Descarregar a *Toolchain* do seguinte endereço:

<http://blackfin.uclinux.org/gf/download/frsrelease/392/5080/blackfin-toolchain-elf-gcc-4.1-08r1.5-14.i386.tar.bz2>

Guardá-lo numa diretoria desejada.

2. Abrir a consola e navegar até à diretoria onde foi guardado o ficheiro comprimido com a *toolchain*.
3. Uma vez na diretoria da *toolchain* correr o seguinte comando para efetuar a descompactação:

```
sudo tar xvfj blackfin-toolchain-elf-gcc-4.1-08r1.5-14.i386.tar.bz2
```

E assim se conclui a instalação da *toolchain*. Já é possível passar à compilação.

## 2. Compilação do *firmware*

1. Descarregar o ficheiro com o *firmware* do endereço:

<http://code.google.com/p/surveyor-srv1-firmware/downloads/list>

Selecionar a versão mais recente.

2. Descompactá-lo e guarda-lo numa diretoria desejada.

3. Agora é necessário adicionar a diretoria da *toolchain* à variável “PATH” do sistema. Neste caso usou-se o seguinte comando:

```
sudo export PATH=$PATH:/home/rodolfo/opt/uClinux/bfin-elf/bin/
```

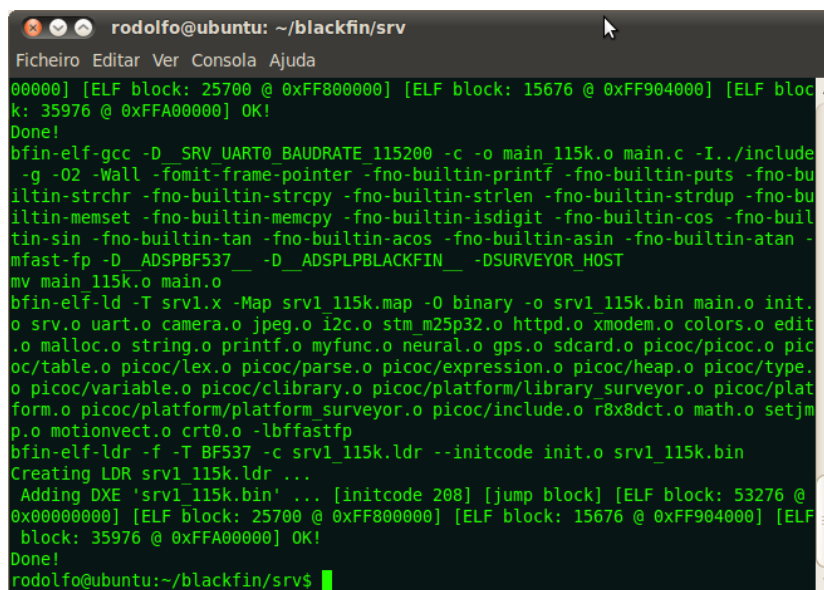
Não esquecer de substituir a diretoria “/home/rodolfo/opt/uClinux/bfin-elf/bin/” pela diretoria onde o ficheiro da *toolchain* foi descompactado.

4. Abrir a consola e navegar até onde o ficheiro com o *firmware* foi descompactado.

5. Correr o seguinte comando:

```
make
```

Deverá ser avistada a seguinte mensagem:



```
rodolfo@ubuntu: ~/blackfin/srv
Ficheiro Editar Ver Consola Ajuda
000000] [ELF block: 25700 @ 0xFF800000] [ELF block: 15676 @ 0xFF904000] [ELF block: 35976 @ 0xFFA00000] OK!
Done!
bfin-elf-gcc -D _SRV_UART0_BAUDRATE_115200 -c -o main_115k.o main.c -I./include
-g -O2 -Wall -fomit-frame-pointer -fno-builtin-printf -fno-builtin-puts -fno-bu
iltin-strchr -fno-builtin-strcpy -fno-builtin-strlen -fno-builtin-strdup -fno-bu
iltin-memset -fno-builtin-memcpy -fno-builtin-isdigit -fno-builtin-cos -fno-buil
tin-sin -fno-builtin-tan -fno-builtin-acos -fno-builtin-asin -fno-builtin-atan -
mfast-fp -D _ADSPBF537__ -D _ADSPLBLACKFIN__ -DSURVEYOR_HOST
mv main_115k.o main.o
bfin-elf-ld -T srv1.x -Map srv1_115k.map -O binary -o srv1_115k.bin main.o init.
o srv.o uart.o camera.o jpeg.o i2c.o stm_m25p32.o httpd.o xmodem.o colors.o edit
.o malloc.o string.o printf.o myfunc.o neural.o gps.o sdcard.o picoc/picoc.o pic
oc/table.o picoc/lex.o picoc/parse.o picoc/expression.o picoc/heap.o picoc/type.
o picoc/variable.o picoc/clibrary.o picoc/platform/library_surveyor.o picoc/plat
form.o picoc/platform/platform_surveyor.o picoc/include.o r8x8dct.o math.o setj
p.o motionvect.o crt0.o -lbffastfp
bfin-elf-ldr -f -T BF537 -c srv1_115k.ldr --initcode init.o srv1_115k.bin
Creating LDR srv1_115k.ldr ...
Adding DXE 'srv1_115k.bin' ... [initcode 208] [jump block] [ELF block: 53276 @
0x00000000] [ELF block: 25700 @ 0xFF800000] [ELF block: 15676 @ 0xFF904000] [ELF
block: 35976 @ 0xFFA00000] OK!
Done!
rodolfo@ubuntu:~/blackfin/srv$
```

Figura A.1 - Mensagem recebida após compilação.

### 3. Instalação do *Moserial*

1. Abrir a consola e correr o seguinte comando para efetuar a instalação do Moserial:

`sudo apt-get install Moserial`

2. Agora é necessário instalar uma *package* que permite efetuar as transferências através do protocolo XMODEM. Para isso correr o comando:

`sudo apt-get install lrzsz`

Pode-se agora executar o programa *Moserial* que tem o seguinte aspeto:



Figura A.2 - Aspeto do Moserial.

### 4. Transferir o ficheiro *srv1.ldr*

1. Abrir o programa *Moserial*.
2. Ir às configurações.



Figura A.3 - Configurações.

3. Selecionar a porta COM à qual o sistema está conectado, selecionar a *baudrate* presente no *firmware*, configurar os restantes campos como mostra a figura.

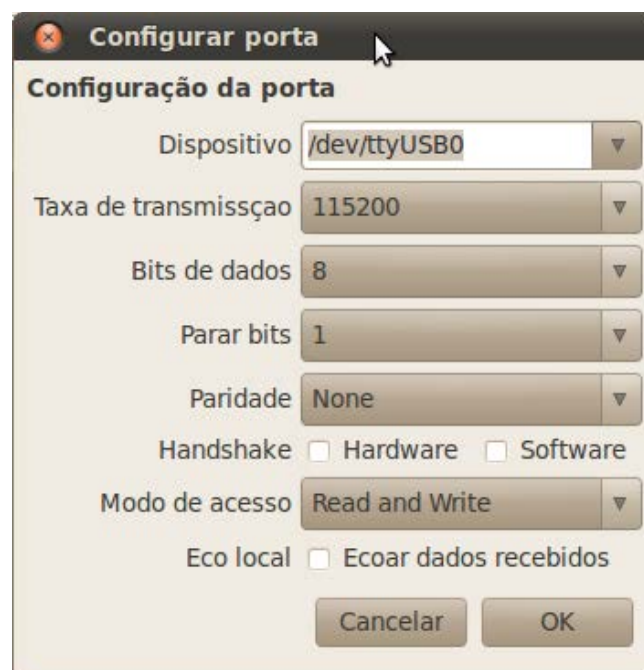


Figura A.4 - Configuração da porta série.

4. Enviar o caractere “V” para verificar se a ligação está bem estabelecida. Se for recebida a versão do *firmware* a conexão foi feita com sucesso.

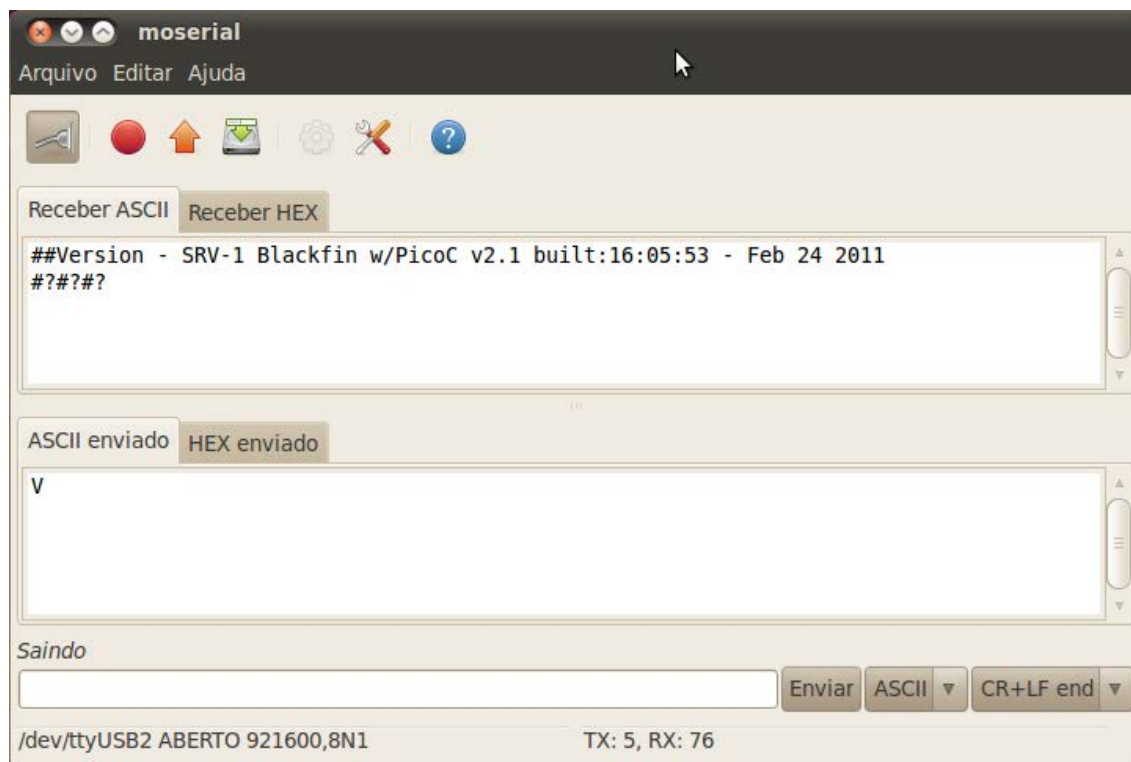


Figura A.5 - Versão do firmware.

5. Enviar o comando "X", deverá ser recebido o caractere "C".
6. Carregar em envio de arquivos.

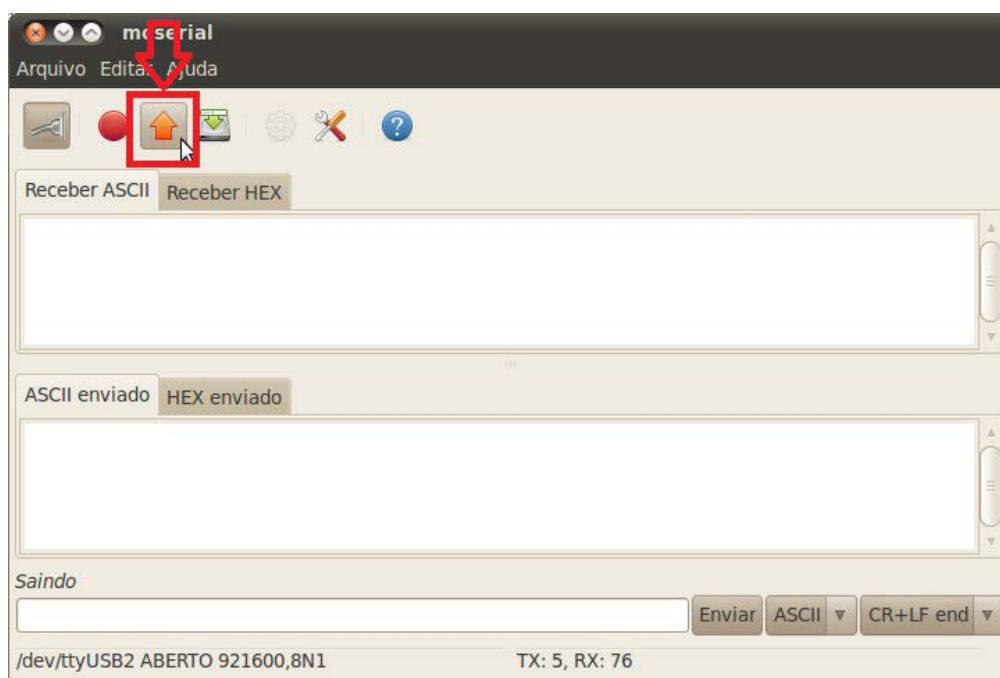


Figura A.6 - Envio do arquivo.

7. Selecionar o ficheiro "srv1.ldr" (resultado da compilação) e o protocolo de correção de erro XMODEM.

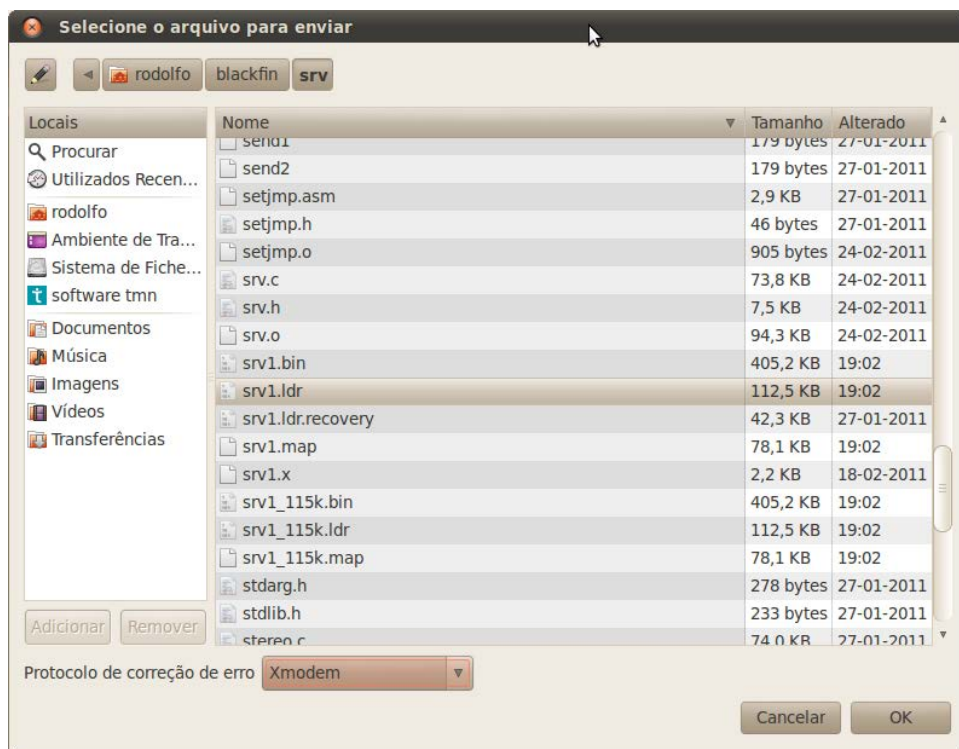


**Figura A.7 - Seleção do arquivo.**

- Depois de concluída a transferência enviar o comando “zZ”. Depois de recebida a confirmação da imagem nas zonas de memória reiniciar a câmara.

E assim se conclui este tutorial.

## Anexo B

### Compilação e Transferência do *firmware* em Windows

Este tutorial tem como objetivo explicar os vários procedimentos necessários para compilar o *firmware* e transferir o ficheiro resultante para a *smart camera*.

Para poder-se compilar o *firmware* disponibilizado pelo fabricante é necessário ter os seguintes *softwares* instalados no computador:

- Mingw32 - Que é um conjunto de ferramentas GNU portadas para Windows que nos permite fazer a compilação de programas C em sistemas operativos Windows.
- TeraTerm - Permite efetuar ligações pela porta série. Neste tutorial não haverá explicação de instalação visto ser um programa de instalação *standard*.
- Blackfin toolchain - Contém as bibliotecas necessárias para a compilação do *firmware*.

#### 1. Instalação do Mingw32

1. Descarregar o executável do endereço: <http://sourceforge.net/projects/mingw/>
2. Correr o executável.

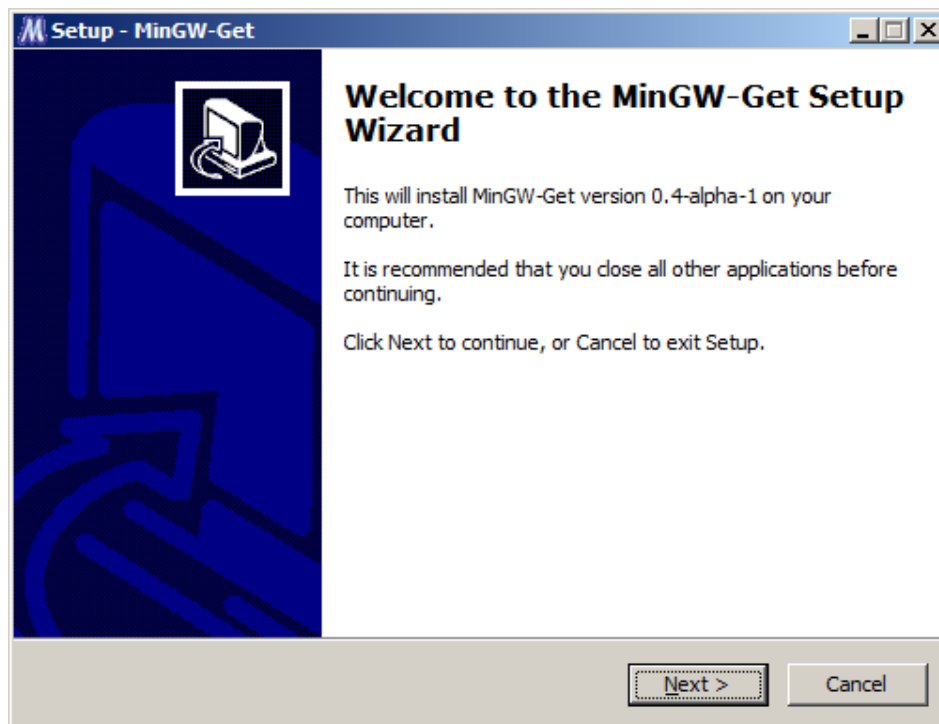


Figura B.1 - Instalação Mingw32 1.

Clicar em “Next>”.

3. Selecionar a opção “*Download latest repository catalogues*” e clicar em “Next”.

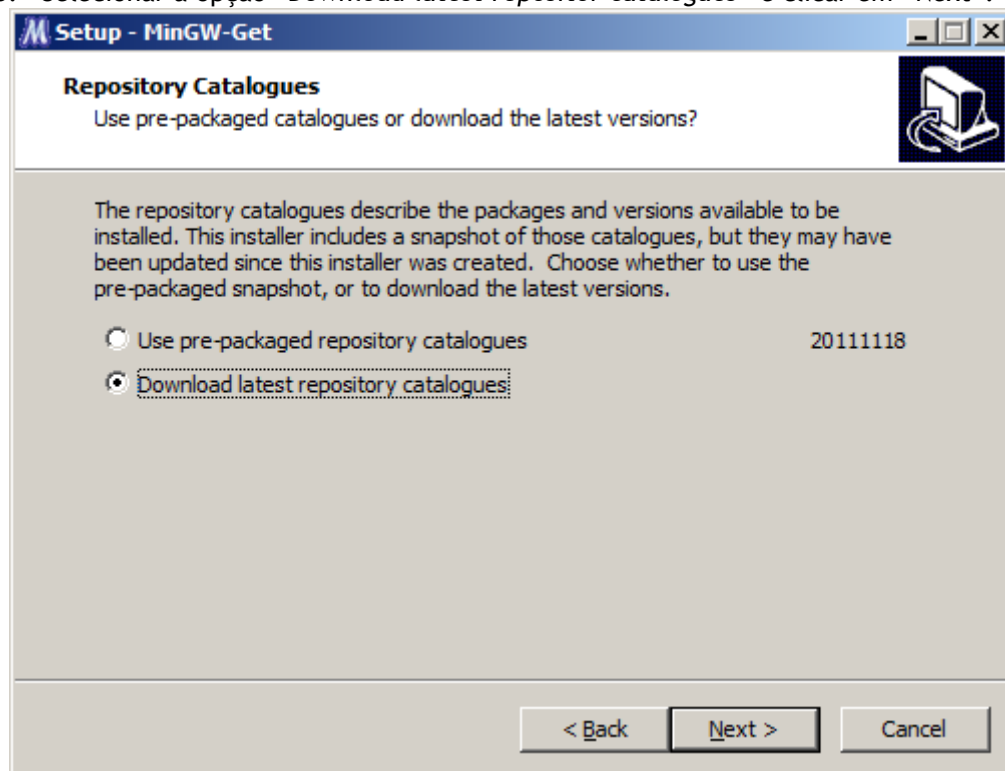


Figura B.2 - Instalação Mingw32 2.

4. Ler e aceitar e depois clicar em “Next”.

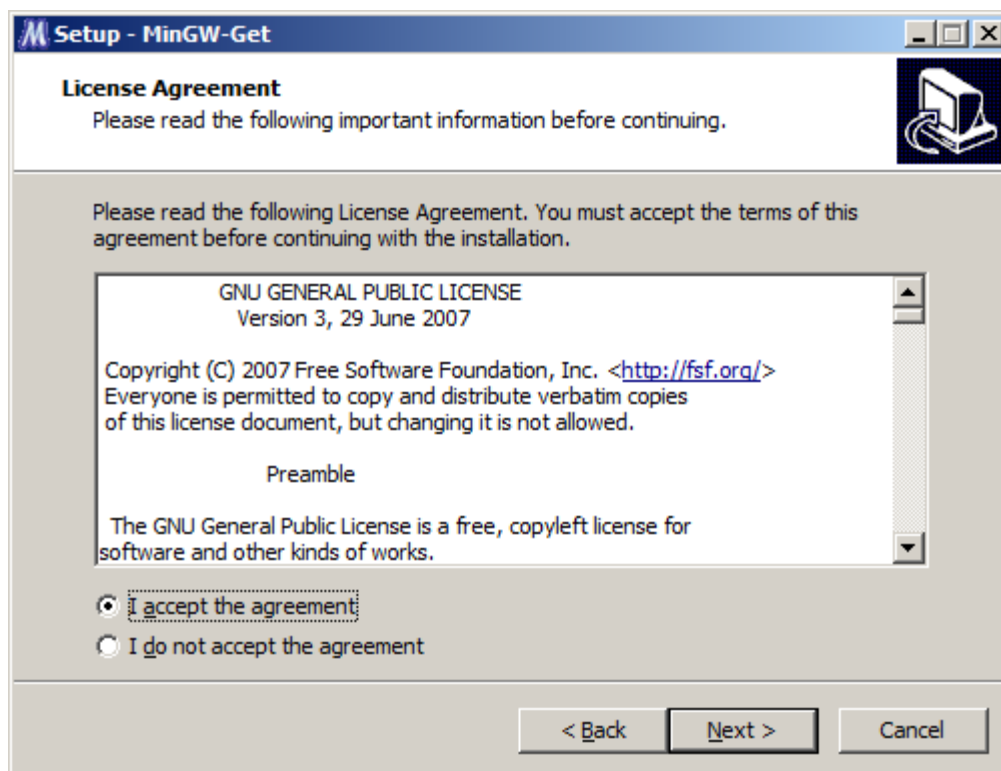


Figura B.3 - Instalação Mingw32 3.

5. Selecionar a pasta de instalação (recomenda-se a diretoria padrão) e clicar “Next”.

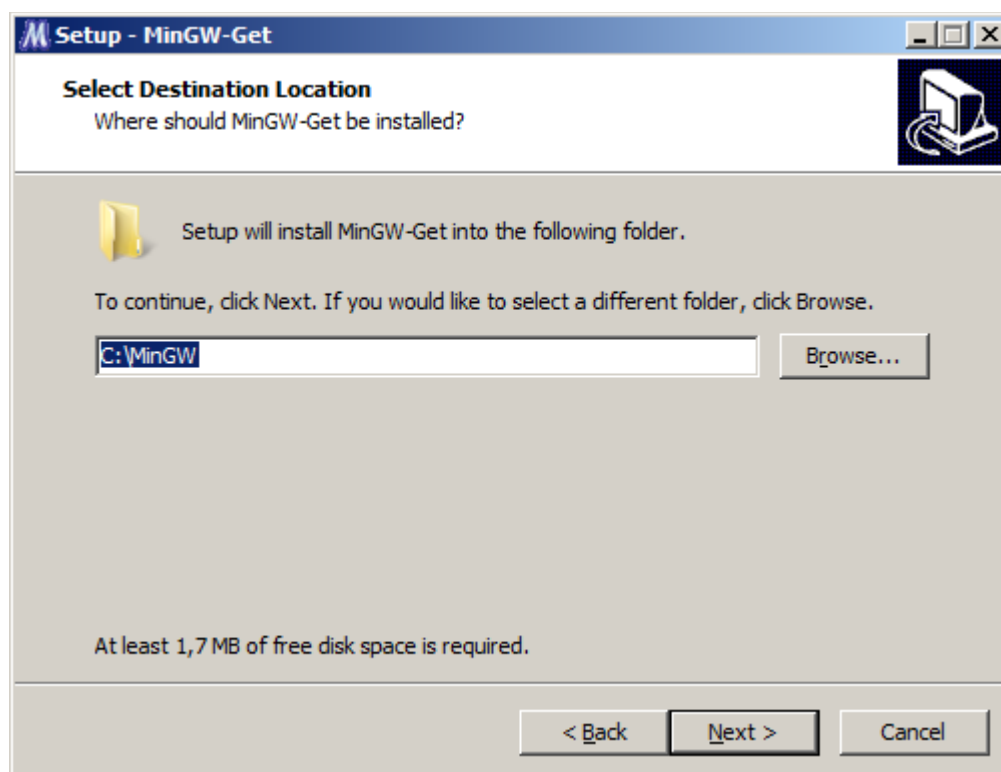


Figura B.4 - Instalação Mingw32 4.

6. Como o *firmware* do fabricante está escrito em C seleciona-se apenas o compilador C.

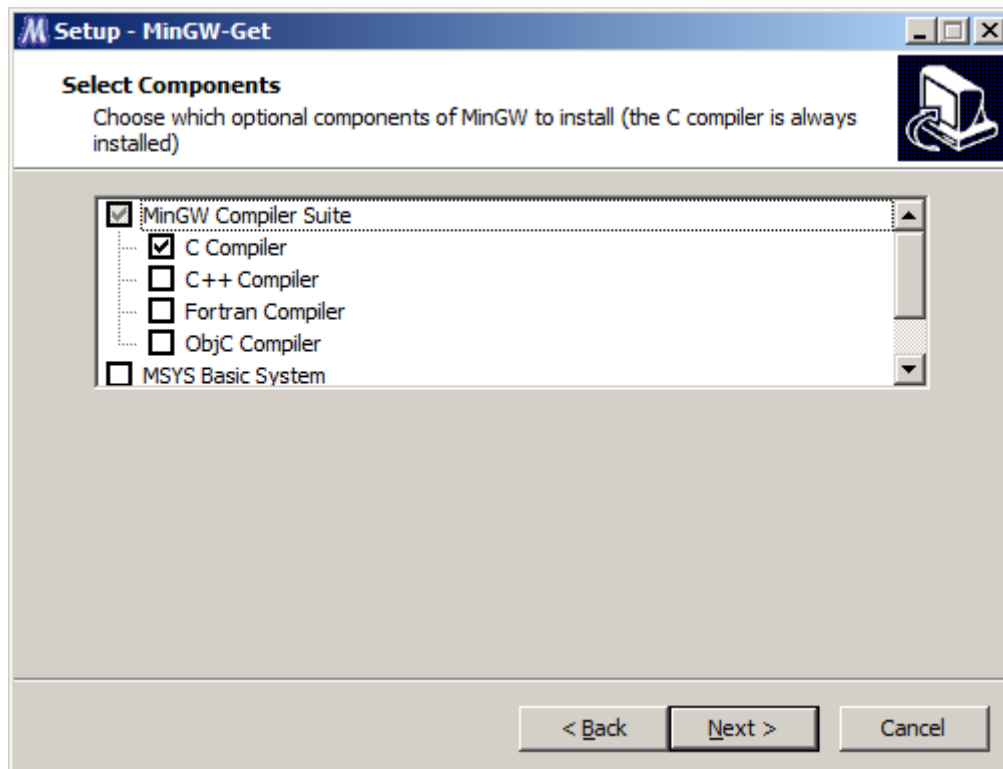


Figura B.5 - Instalação Mingw32 5.

7. Finalmente clicar *install* e esperar que os *updates*, *downloads* e instalação sejam executados.

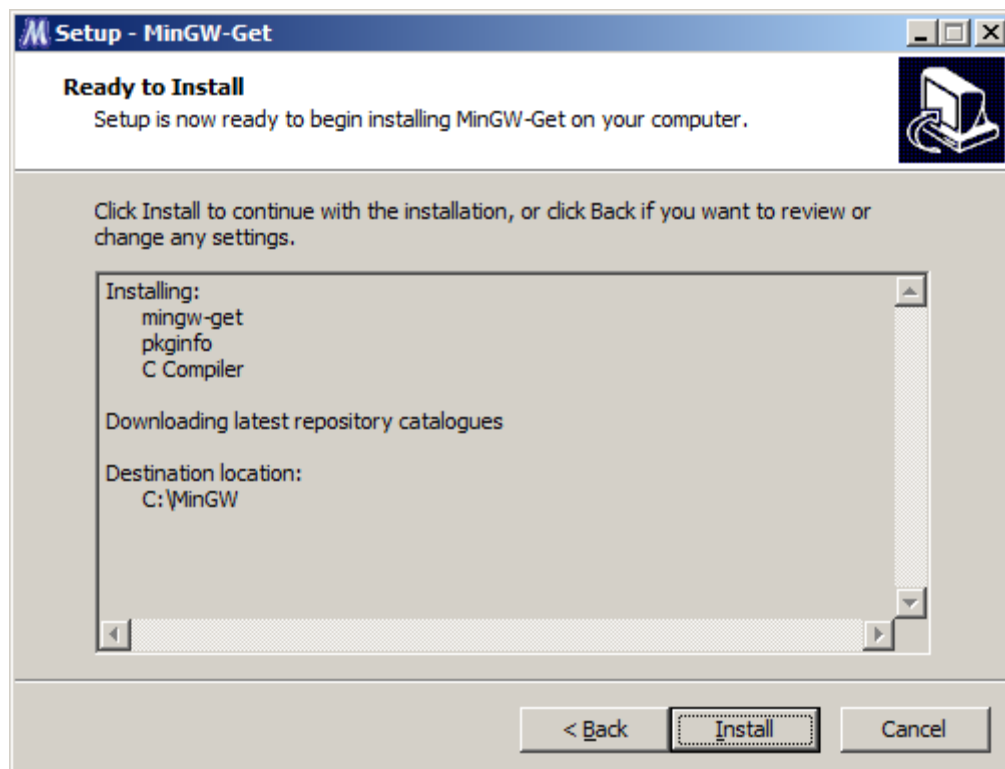


Figura B.6 - Instalação Mingw32 6.

## 2. Instalação da Blackfin toolchain

1. Descarregar o ficheiro de instalação do seguinte endereço:  
<http://blackfin.uclinux.org/gf/download/frsrelease/392/5211/blackfin-toolchain-win32-2008R1.5.exe>
2. Correr o ficheiro.
3. Carregar em “Next”.

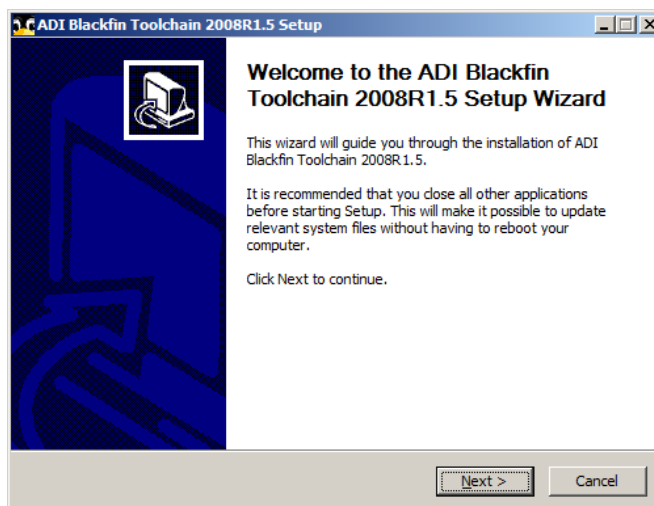


Figura B.7 - Instalação Toolchain 1.

4. Ler, aceitar e carregar “Next”.

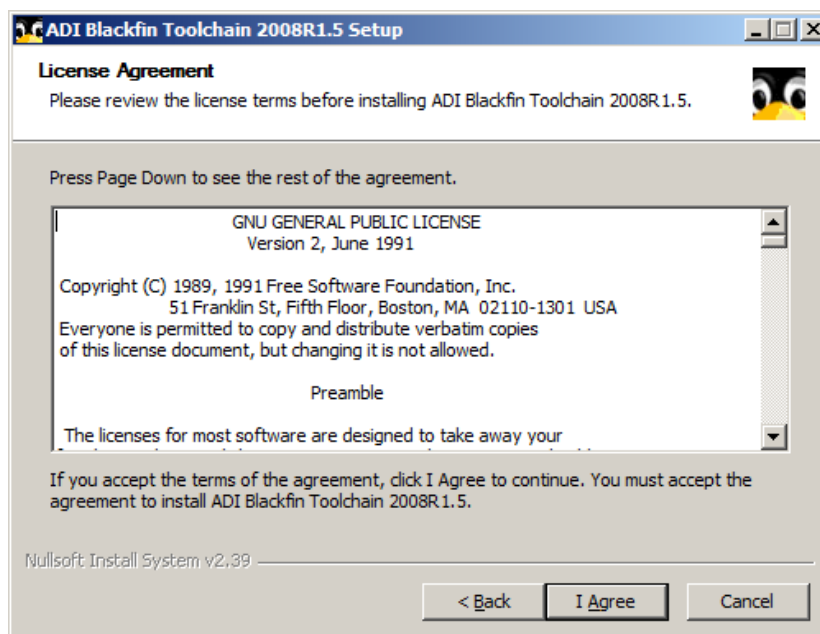


Figura B.8 - Instalação Toolchain 2.

5. Selecionar todos os componentes e clicar “Next”.

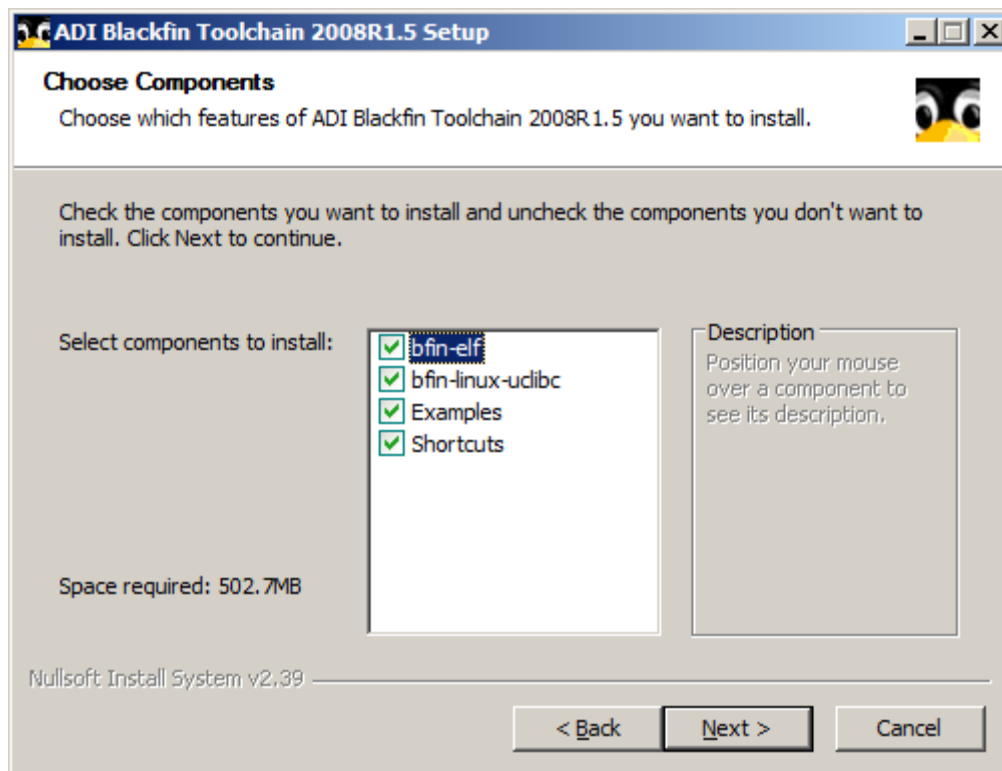


Figura B.9 - Instalação *Toolchain 3*.

6. Selecionar a diretoria de instalação e clicar “Install”.

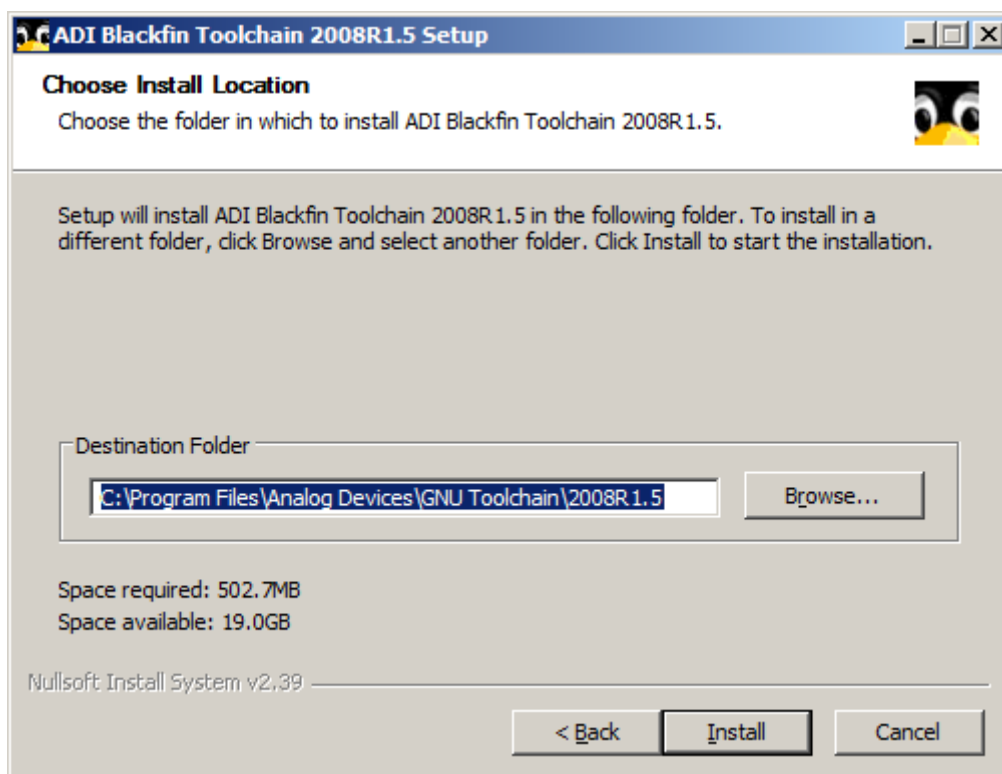


Figura B.10 - Instalação *Toolchain 4*.

**IMPORTANTE:** Quando aparecer a seguinte janela no ecrã clicar em **Não**. Se seleccionar sim irão haver conflitos com as portas USB do computador.

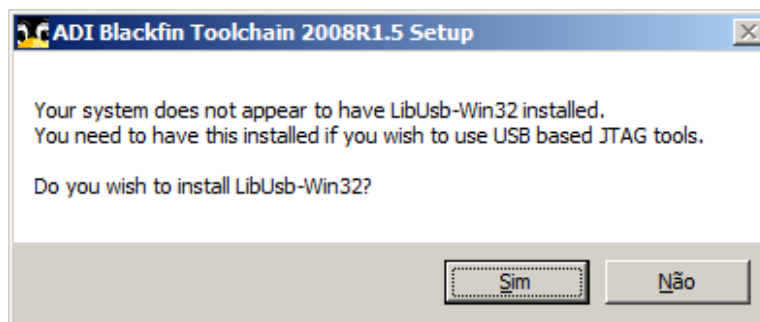


Figura B.11 - Instalação *Toolchain* 5.

Nesta altura tem-se os dois programas necessários instalados.

### 3. Compilação do *firmware*

#### 1. Adicionar à variável “path” o diretório da toolchain

Primeiro é necessário adicionar à variável “path” do sistema operativo o local onde estão armazenados os ficheiros necessários à compilação. Desta forma o sistema saberá onde procurar as bibliotecas. No sistema operativo Windows 7 (nas outras versões é muito similar) faz-se da seguinte maneira.

1. Clique com o botão direito do rato no ícone “O meu computador”.
2. Clicar em “Propriedades”.
3. Clicar em “Definições avançadas do sistema”.
4. Clicar em “Variáveis de Ambiente”.

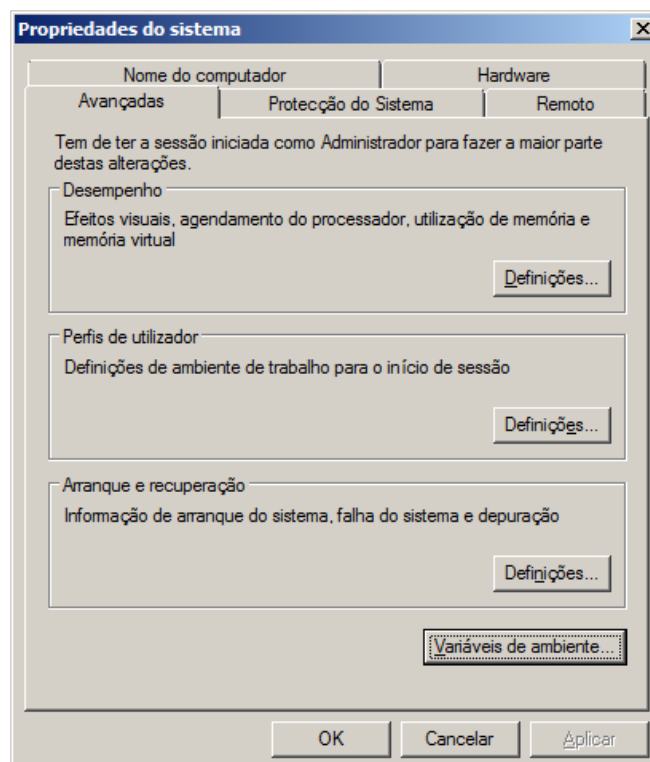


**Figura B.12 - Propriedades do sistema**

5. Em variáveis do sistema seleccionar a variável “PATH” e clicar em “Editar”.

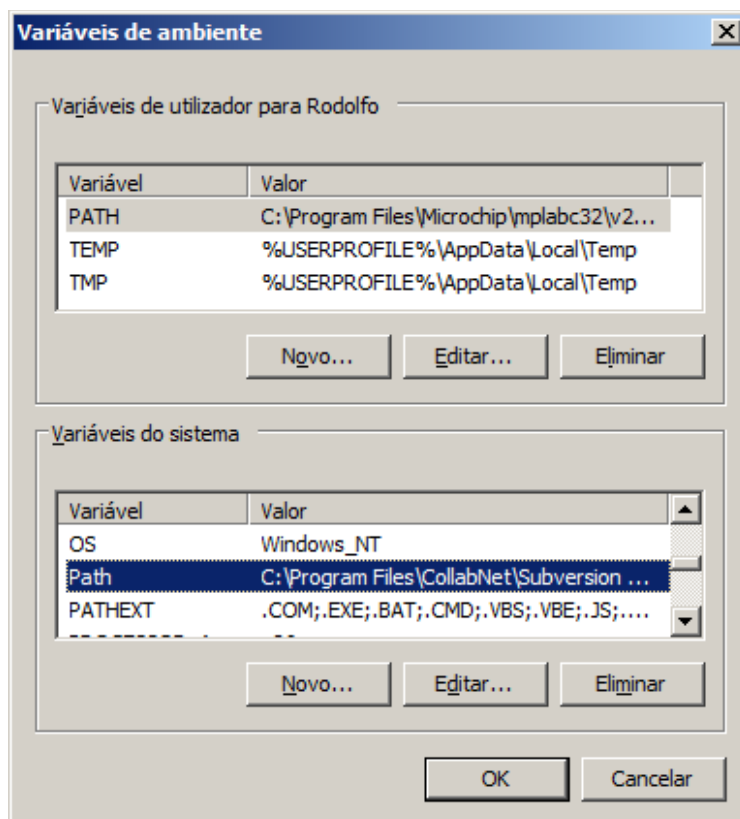


**Figura B.13 - Variáveis de ambiente.**

- Adicionar a diretória onde se encontra instalada a *Blackfin toolchain* à variável do sistema. No caso do computador presente a diretoria foi:

C:\Program Files\Analog Devices\GNU Toolchain\2008R1.5\linux-uclibc\bin

Usar o caracter “;” para separar das diretorias já existentes como indica a figura seguinte:

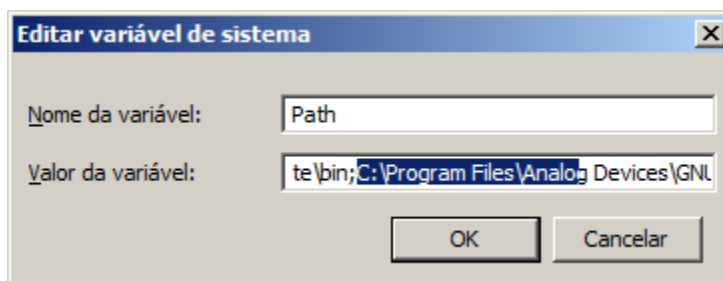


Figura B.14 - Edição de variável do sistema.

E assim se conclui o processo de alteração da variável “Path”.

## 2. Copiar o executável “mingw32-make.exe” para a diretoria onde se encontra o *firmware*

- Descarregar o *firmware* mais recente do seguinte endereço:  
<http://code.google.com/p/surveyor-srv1-firmware/downloads/list>
- Descompactar o *firmware* numa diretoria desejada.
- Ir à diretoria onde foi instalado o Mingw32 e copiar o ficheiro “mingw32-make.exe”.
- Colar o ficheiro anteriormente copiado na pasta “srv” da diretoria onde foi descompactado o *firmware*.
- Sugere-se que seja alterado o nome do ficheiro colado para “make.exe”.

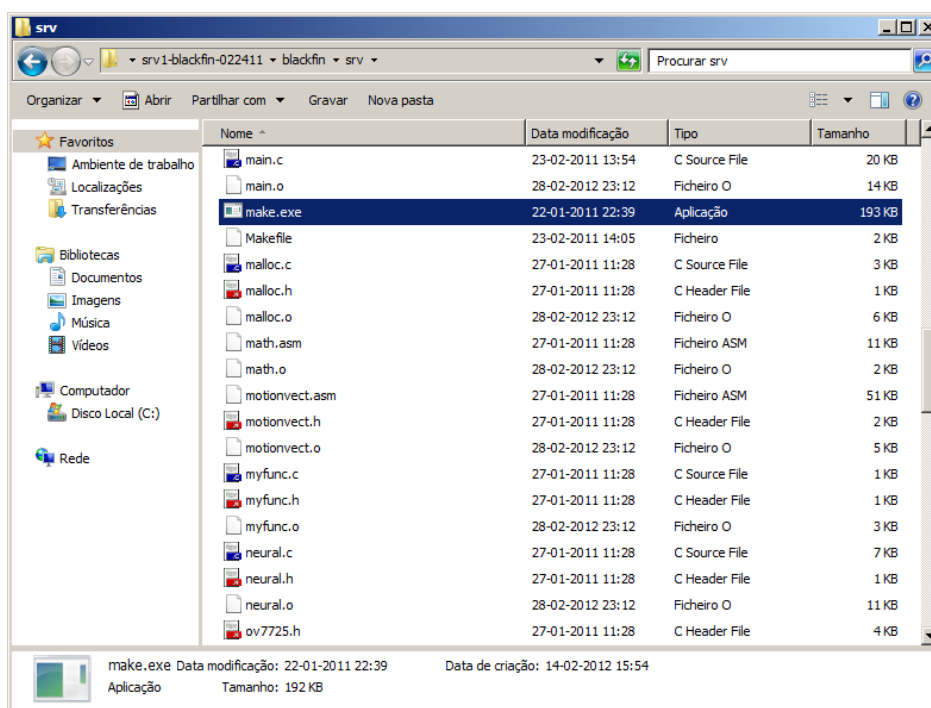
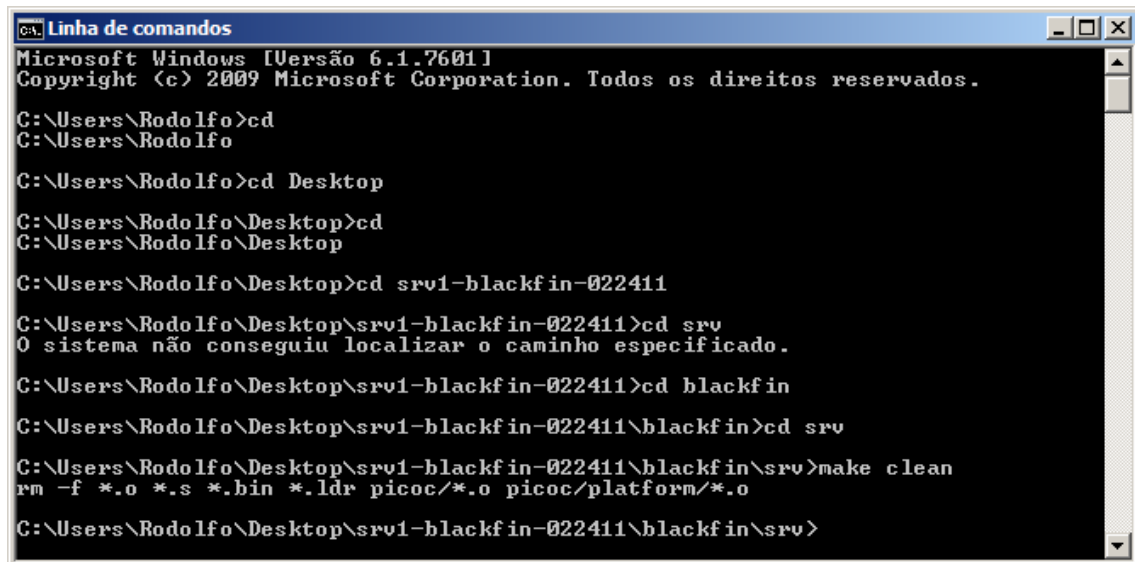


Figura B.15 - Compilação 1.

### 3. Compilação do *firmware*

Nesta fase já está tudo configurado para compilar o *firmware*.

1. Abrir a linha de comandos.
2. Navegar até à pasta “srv” onde o *firmware*” foi descompactado.
3. Introduzir o comando “make clean”.



```
Microsoft Windows [Versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Rodolfo>cd
C:\Users\Rodolfo

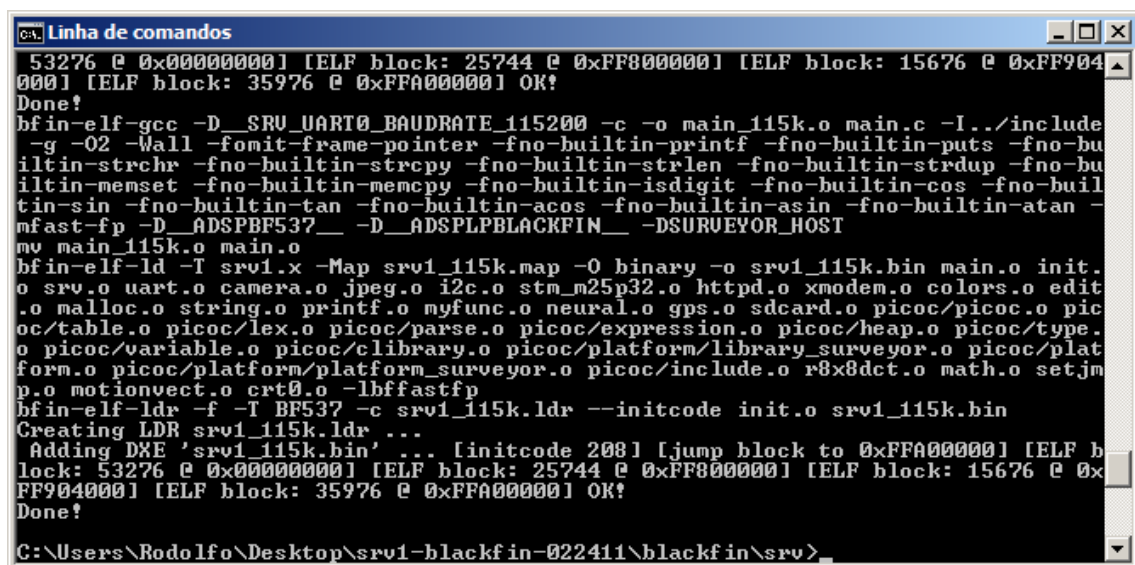
C:\Users\Rodolfo>cd Desktop
C:\Users\Rodolfo\Desktop>cd
C:\Users\Rodolfo\Desktop

C:\Users\Rodolfo\Desktop>cd srv1-blackfin-022411
C:\Users\Rodolfo\Desktop\srv1-blackfin-022411>cd srv
O sistema não conseguiu localizar o caminho especificado.

C:\Users\Rodolfo\Desktop\srv1-blackfin-022411>cd blackfin
C:\Users\Rodolfo\Desktop\srv1-blackfin-022411\blackfin>cd srv
C:\Users\Rodolfo\Desktop\srv1-blackfin-022411\blackfin\srv>make clean
rm -f *.o *.s *.bin *.ldr picoc/*.o picoc/platform/*.o
C:\Users\Rodolfo\Desktop\srv1-blackfin-022411\blackfin\srv>
```

Figura B.16 - Compilação 2.

4. Introduzir o comando “make”. O *firmware* é então compilado. Como resultado da compilação iremos ter o ficheiro “srv1.ldr” que depois será transferido para o sistema. Ter em conta que é muito importante verificar a *baudrate* definida no *firmware*. Desta forma será conhecida a *baudrate* com que deve ser feita a ligação via porta série. Para fazer esta verificação ir ao ficheiro *config.h* do *firmware*.



```
53276 @ 0x00000000] [ELF block: 25744 @ 0xFF800000] [ELF block: 15676 @ 0xFF904000] [ELF block: 35976 @ 0xFFA00000] OK!
Done!
bfin-elf-gcc -D__SRV_UART0_BAUDRATE_115200 -c -o main_115k.o main.c -I../include
-g -O2 -Wall -fomit-frame-pointer -fno-builtin-printf -fno-builtin-puts -fno-bu
iltin-strchr -fno-builtin-strcpy -fno-builtin-strlen -fno-builtin-strdup -fno-bu
iltin-memset -fno-builtin-memcpy -fno-builtin-isdigit -fno-builtin-cos -fno-buil
tin-sin -fno-builtin-tan -fno-builtin-acos -fno-builtin-asin -fno-builtin-atan -
mfast-fp -D__ADSPBF537__ -D__ADSPBLACKFIN__ -DSURVEYOR_HOST
mv main_115k.o main.o
bfin-elf-ld -T srv1.x -Map srv1_115k.map -O binary -o srv1_115k.bin main.o init.
o srv.o uart.o camera.o jpeg.o i2c.o stm_m25p32.o httpd.o xmodem.o colors.o edit
.o malloc.o string.o printf.o myfunc.o neural.o gps.o sdcard.o picoc/picoc.o pic
oc/table.o picoc/lex.o picoc/parse.o picoc/expression.o picoc/heap.o picoc/type.
o picoc/variable.o picoc/clibrary.o picoc/platform/library_surveyor.o picoc/plat
form.o picoc/platform/platform_surveyor.o picoc/include.o r8x8dct.o math.o setjm
p.o motionvect.o crt0.o -lbffastfp
bfin-elf-ldr -f -T BF537 -c srv1_115k.ldr --initcode init.o srv1_115k.bin
Creating LDR srv1_115k.ldr ...
Adding DEX 'srv1_115k.bin' ... [initcode 208] [jump block to 0xFFA00000] [ELF b
lock: 53276 @ 0x00000000] [ELF block: 25744 @ 0xFF800000] [ELF block: 15676 @ 0x
FF904000] [ELF block: 35976 @ 0xFFA00000] OK!
Done!
C:\Users\Rodolfo\Desktop\srv1-blackfin-022411\blackfin\srv>
```

Figura B.17 - Compilação 3.

4. Transferir o ficheiro “*srv1.ldr*” para o *flash buffer*.

1. Abrir o *TeraTerm*.
2. Clicar em “*Setup*” na barra horizontal e em seguida em “*Serial Port*”
3. Selecionar a porta COM à qual o sistema está conectado, selecionar a *baudrate* presente no *firmware*, configurar os restantes campos como mostra a figura.

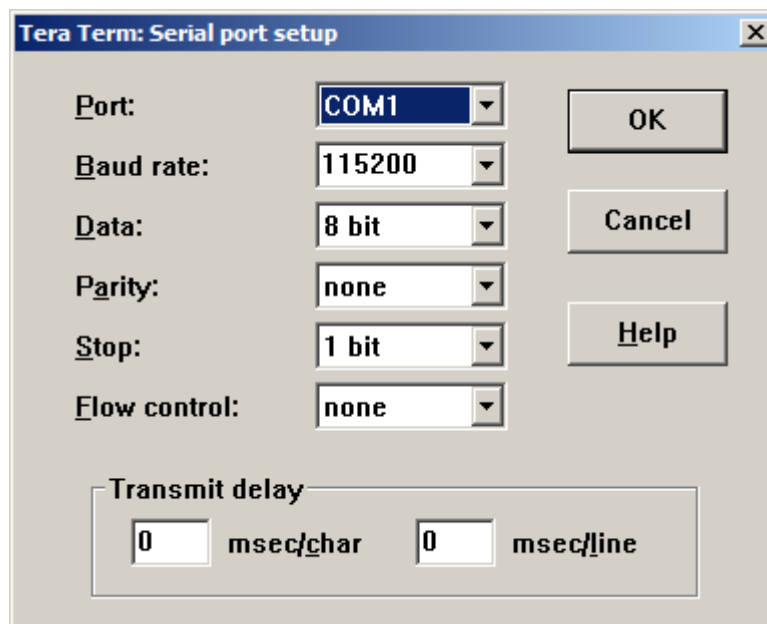


Figura B.18 - Configuração da porta série.

4. Enviar o caractere “V” para verificar se a ligação está bem estabelecida. Se for recebida a versão do *firmware* a conexão foi feita com sucesso.
5. Enviar o comando “X”, deverá ser recebido o caractere “C”.
6. Clicar em “File” -> “XMODEM”-> “Send”.

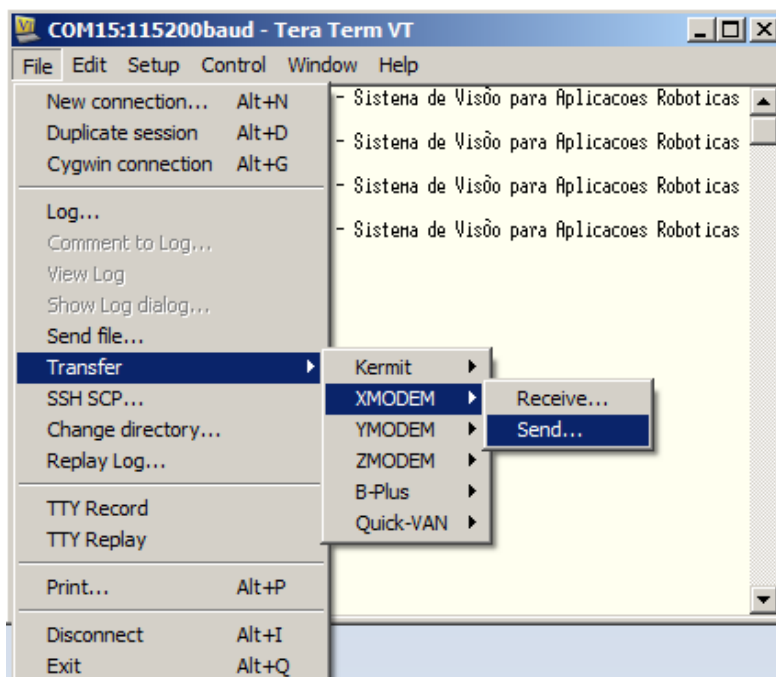


Figura B.19 - Envio do ficheiro loader.

7. Selecionar o ficheiro “srv1.ldr” (resultado da compilação) e clicar em “Abrir”.

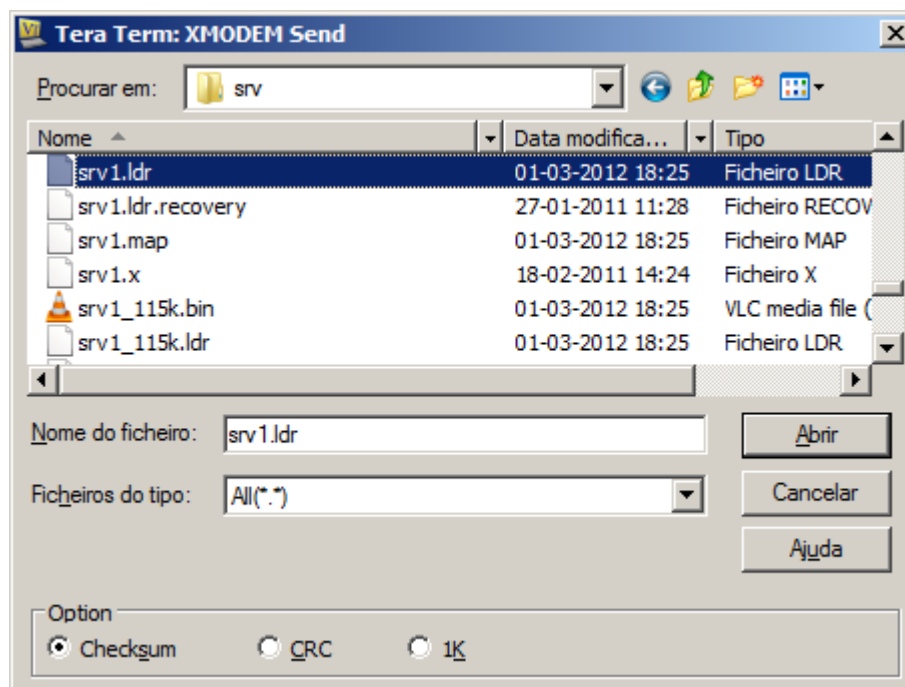


Figura B.20 - Seleção do ficheiro.

8. Depois de recebida uma mensagem de confirmação de transferência sucedida o ficheiro já se encontra no *flash buffer*. Agora é necessário transferir a imagem para o setores de *boot* da memória *flash*.
9. Enviar os seguintes caracteres “Zz”. E desta forma concluímos o processo.

## Anexo C

### Integração da Blackfin Toolchain no ambiente de desenvolvimento Eclipse - Windows

A integração da *Blackfin Toolchain* no *Eclipse* permite uma melhor organização no desenvolvimento da aplicação, um ambiente de desenvolvimento mais “*user-friendly*” e uma melhor eficiência em termos de *debugging*. Para fazer esta integração são necessários os seguintes *softwares*:

- *Blackfin Toolchain* - Esta instalação já foi exemplificada no tutorial “Compilação e transferência do *firmware*”.
- *Eclipse* - Ambiente de desenvolvimento integrado que irá ser utilizada para o desenvolvimento da dissertação. Além do Eclipse serão ainda instalados os *plugins* que irão fazer com que seja possível integrar a *toolchain*.

Neste tutorial apresentam-se todos os passos necessários para a integração da *Toolchain* no Eclipse.

Para que este tutorial possa ser concluído com sucesso é necessário ter efetuado previamente os procedimentos do tutorial “Compilação e transferência do *firmware*”, nomeadamente a instalação da *Blackfin Toolchain* e a configuração da variável “PATH” do sistema.

Neste tutorial foi utilizada a versão *Helios* do Eclipse. Irão ser apresentados exemplos de configuração no sistema operativo Windows mas a configuração em Linux faz-se de forma análoga. Se durante a instalação aparecerem mensagens com avisos de que o *software* a ser instalado não é verificado clicar em prosseguir instalação.

## 1. Instalação de *plugins*

1. Fazer o *download* do Eclipse for C/C++ Developers do seguinte site:

<http://www.eclipse.org/downloads/>

2. Descompactá-lo numa diretoria desejada.
3. Abrir o Eclipse e definir uma diretoria de “workspace” desejada.
4. Na barra de navegação principal, clicar em “Window” -> “Preferences” -> “Install/Update” -> “ Available Software Sites”.

Selecionar o *site*:

<http://download.eclipse.org/tools/cdt/releases/helios>

E clicar em “Enable”.

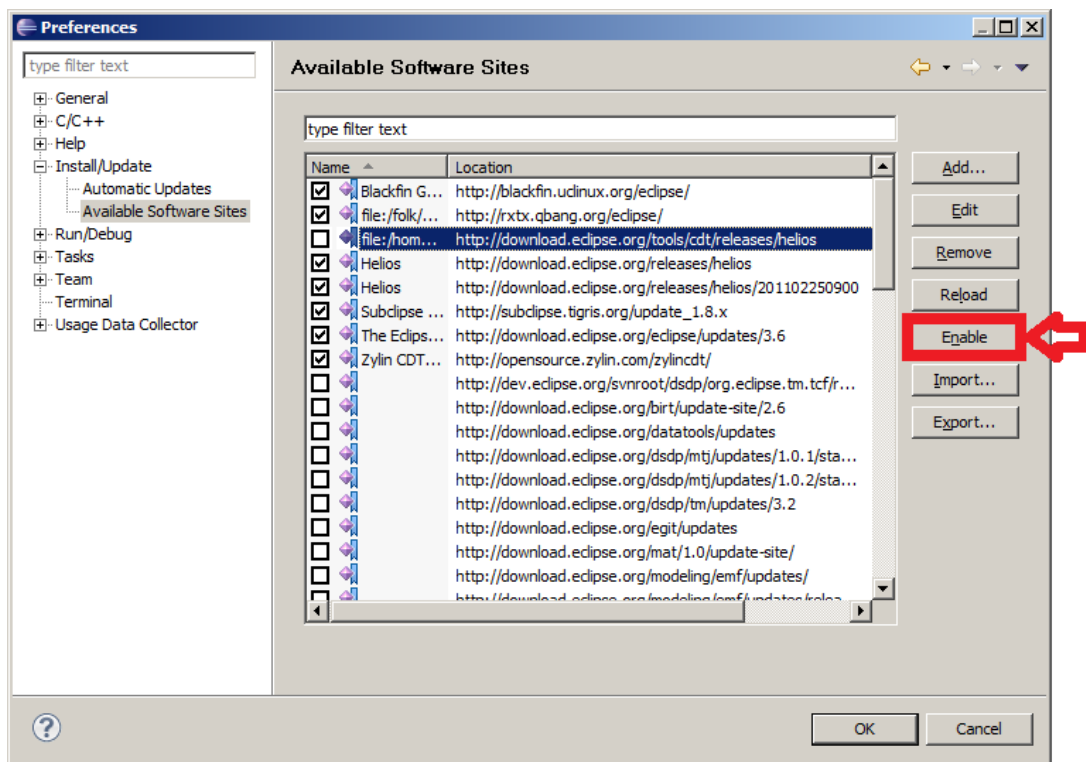


Figura C.1 - Ativação do site.

5. Primeiramente é necessário instalar uma ferramenta necessária para a instalação da *toolchain*.

Na barra de navegação principal clicar em “Help” -> “Install New Software”. Em “Work With” selecionar o *site* referido no ponto 4.

Selecionar todos os componentes a instalar, clicar em “Next”. Concluir a instalação e reiniciar o Eclipse.

Se houver conflitos por causa das versões instalar a versão anterior à última. Para isso fazer *uncheck* a “Show only latest versions of available software”.

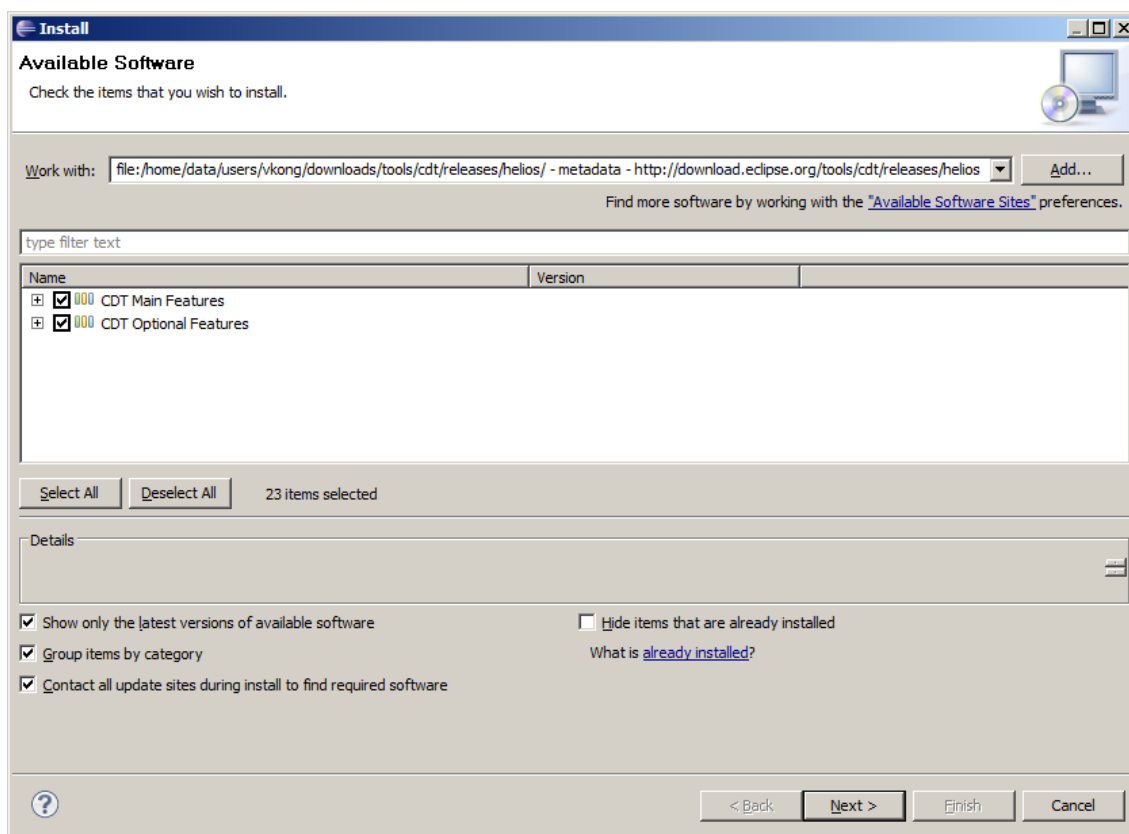


Figura C.2 - Instalação CDT.

6. Passa-se agora à instalação do *plugin* da *toolchain*. Novamente, na barra de navegação principal clicar em “Help” -> “Install New Software”.

Clicar em “Add” e adicionar o seguinte site:

<http://blackfin.uclinux.org/eclipse/>

Em “Work With” selecionar o *site* referido na linha acima.

Selecionar todos os componentes a instalar, clicar em “Next”, concluir a instalação e reiniciar o Eclipse.

Se houver conflitos por causa das versões instalar a versão anterior à última.

Para isso fazer *uncheck* a “Show only latest versions of available software”.

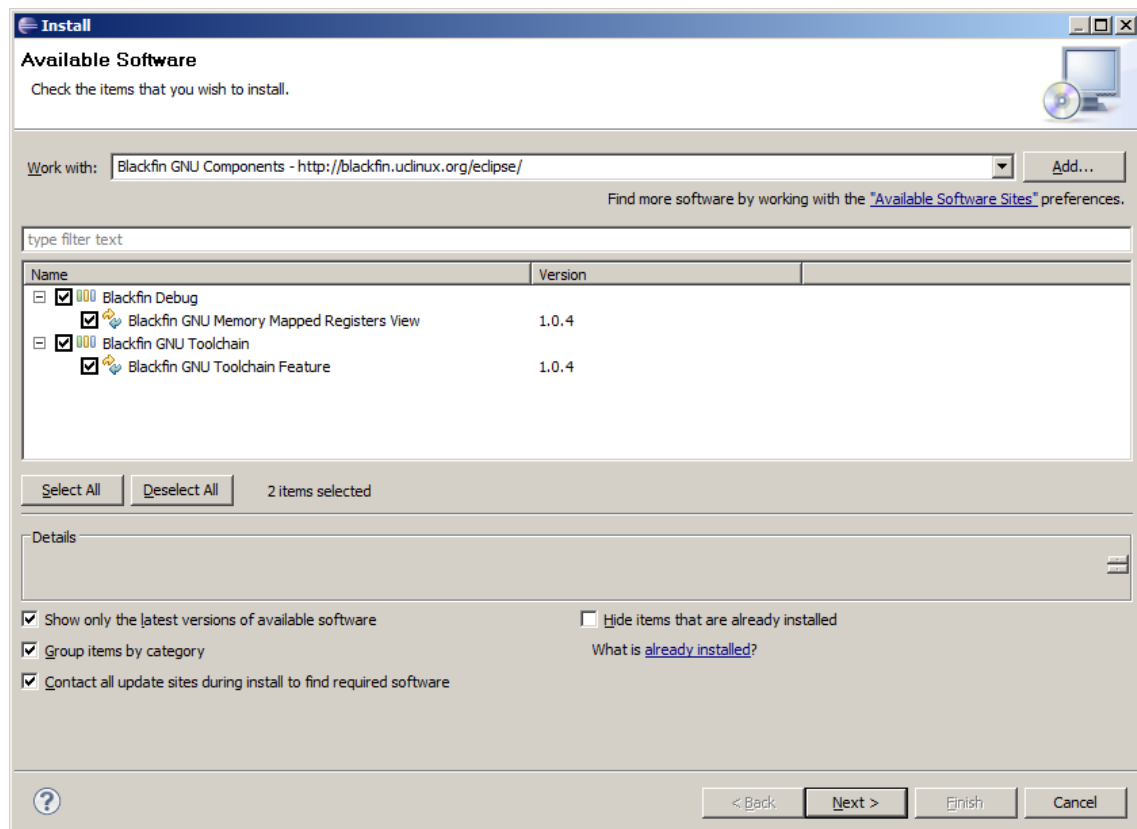


Figura C.3 - Instalação da *Toolchain*.

7. Passa-se agora à instalação de um *plugin* (Zylin CDT) .

Na barra de navegação principal clicar em “Help” -> “Install New Software”.

Clicar em “Add” e adicionar o seguinte *site*:

<http://opensource.zylin.com/zylincdt/>

Em “Work With” selecionar o *site* referido na linha acima.

Selecionar todos os componentes a instalar, clicar em “Next”, concluir a instalação e reiniciar o Eclipse.

Se houver conflitos por causa das versões instalar a versão anterior à última.

Para isso fazer *uncheck* a “Show only latest versions of available software”.

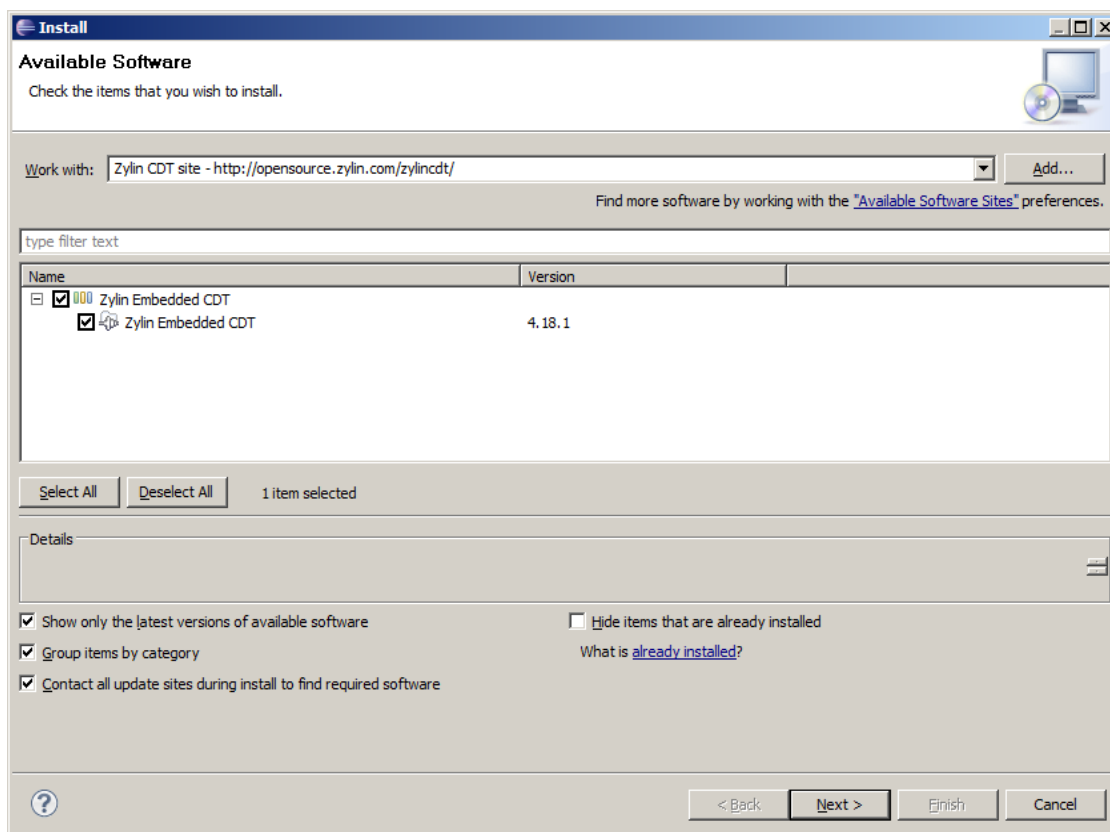


Figura C.4 - Instalação Zylin CDT.

8. Adiciona-se agora um Terminal ao Eclipse para poder-se comunicar com a câmara via porta série.

Repetir o processo de clicar em “Help” -> “Install New Software”.

Em “Work With” seleccionar o *site*:

<http://download.eclipse.org/releases/helios>

Expandir Mobile and Device Development.

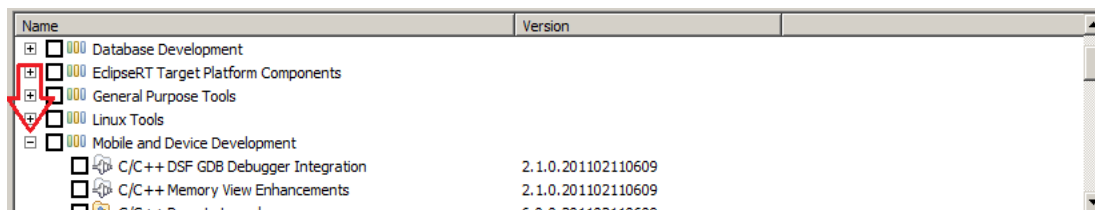


Figura C.5 - Instalação do Terminal.

Selecionar o Target Management Terminal para instalação. Clicar em “Next”, concluir a instalação e reiniciar o Eclipse.

Se houver conflitos por causa das versões instalar a versão anterior à última. Para isso fazer *uncheck* a “Show only latest versions of available software”.

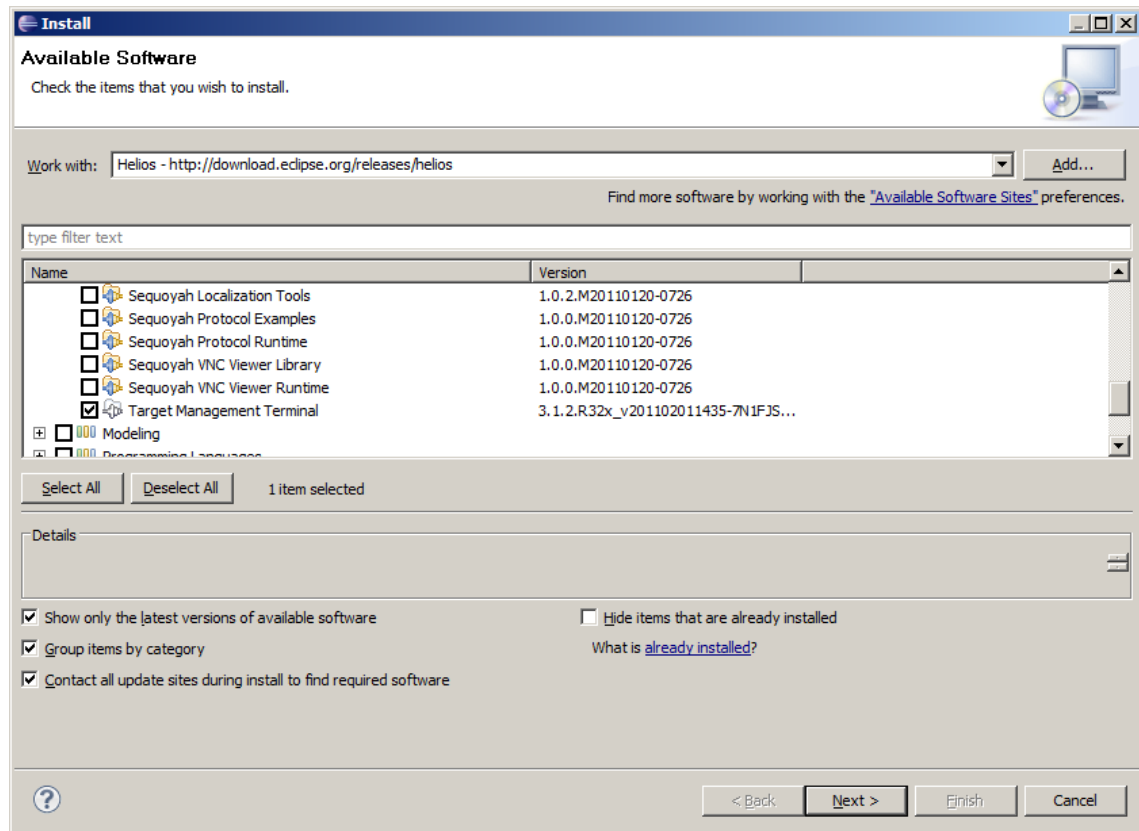


Figura C.6 - Seleção do Terminal para instalação.

9. Para poder-se comunicar via porta série é necessário instalar um *plugin* chamado RXTX.

Na barra de navegação principal clicar em “Help” -> “Install New Software”.

Clicar em “Add” e adicionar o seguinte *site*:

<http://rxtx.qbang.org/eclipse/>

Em “Work With” selecionar o *site* referido na linha acima.

Selecionar a ultima versão clicar em “Next”, concluir a instalação e reiniciar o Eclipse.

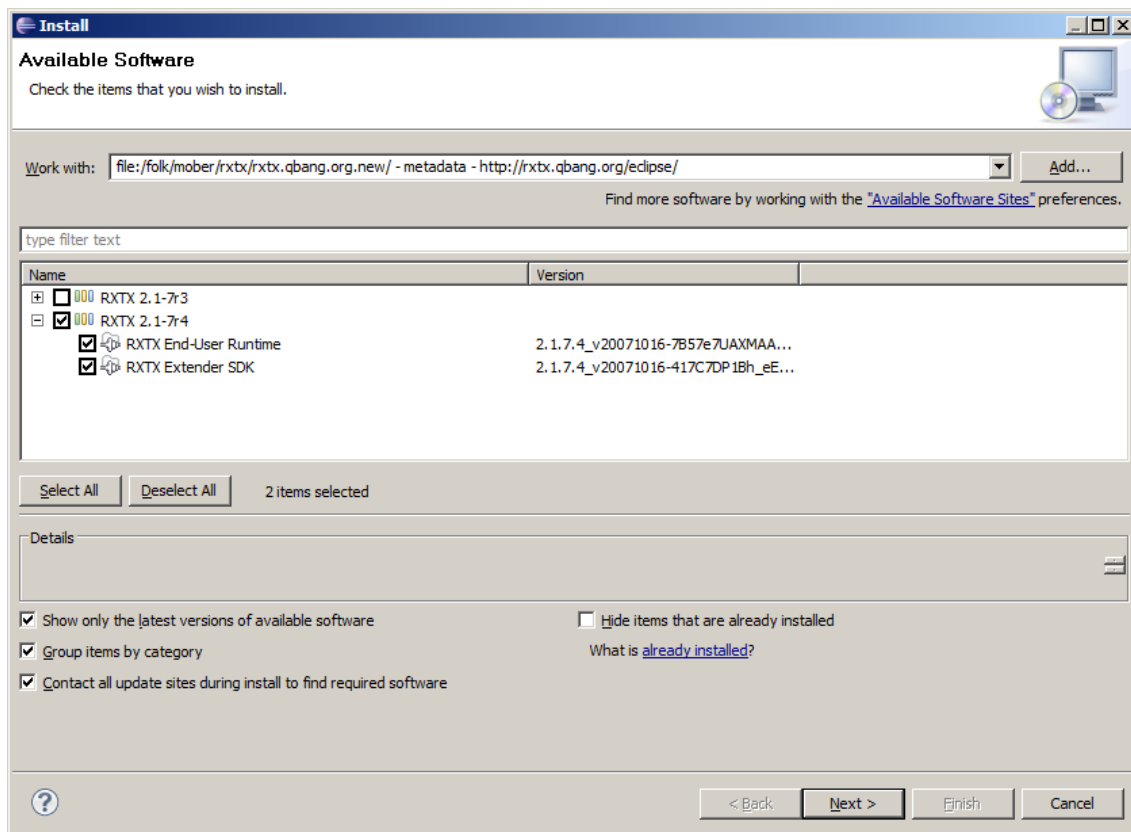


Figura C.7 - Conclusão da instalação do terminal.

E a instalação e configuração dos *plugins* está concluída.

## 2. Criar um novo projeto

1. Abrir o Eclipse.
2. Na barra de navegação principal fazer a seguinte sequência de cliques: “File” - > “New” -> “Project”.
3. Selecionar “Makefile Project with Existing Code” e clicar em “Next”.

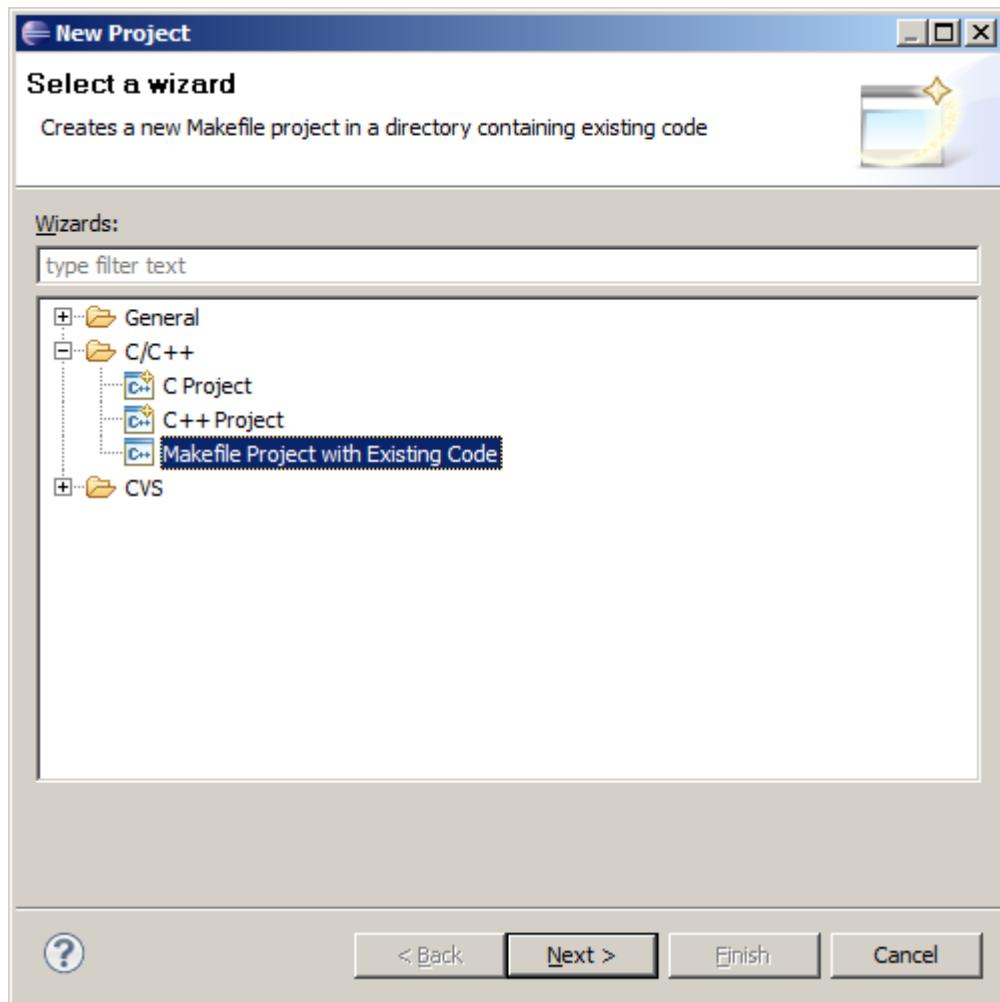


Figura C.8 - Criação de um novo projeto.

4. Dar um nome ao projeto, indicar a localização onde o *firmware* da câmara se encontra e escolher “Blackfin Bare Metal (bfin-elf)” para *toolchain*. Por fim clicar em Finish.

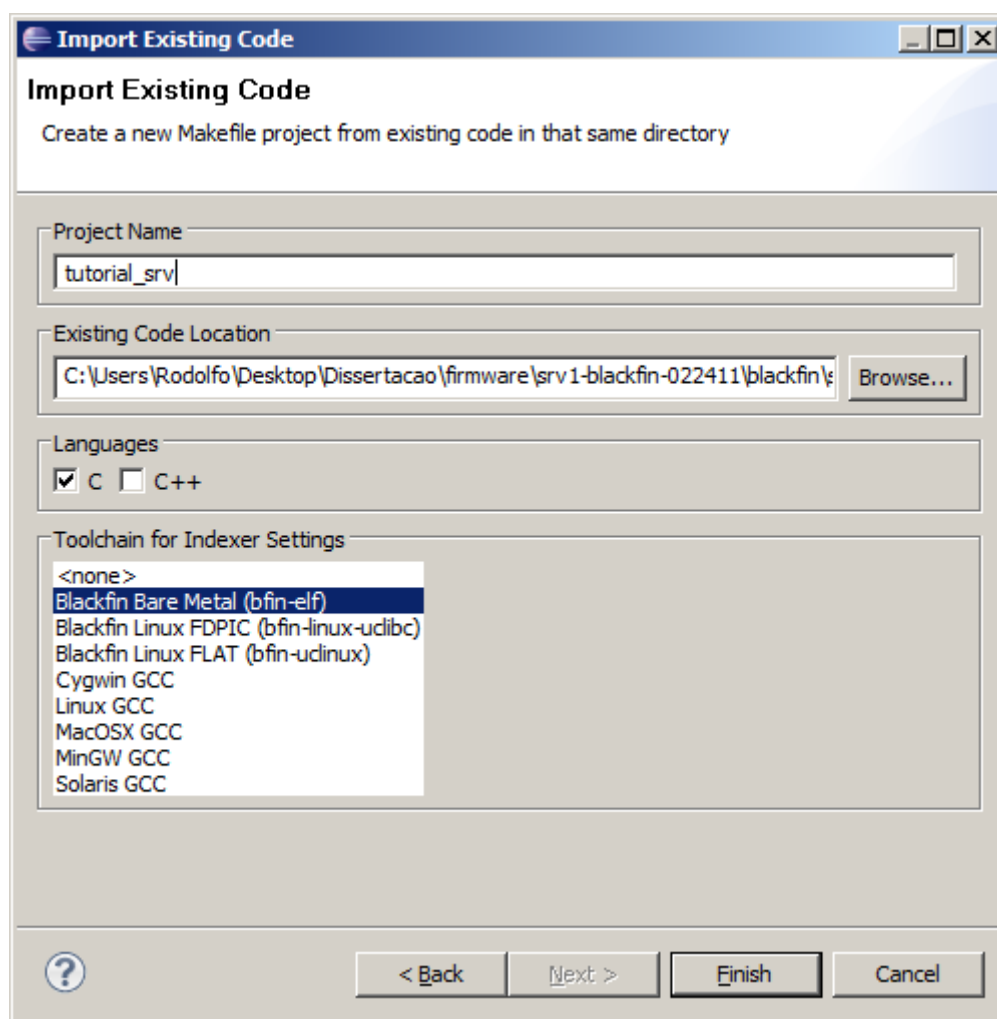


Figura C.9 - Seleção da *Toolchain*.

5. Agora é necessário indicar que processador se está a usar, neste caso o processador conectado à câmara é o BF537. Para isso clicar em “Project” -> “Properties” -> “C/C++ Build” -> “Settings”. Selecionar o processador BF537 e “any” para “Silicon Revision”.

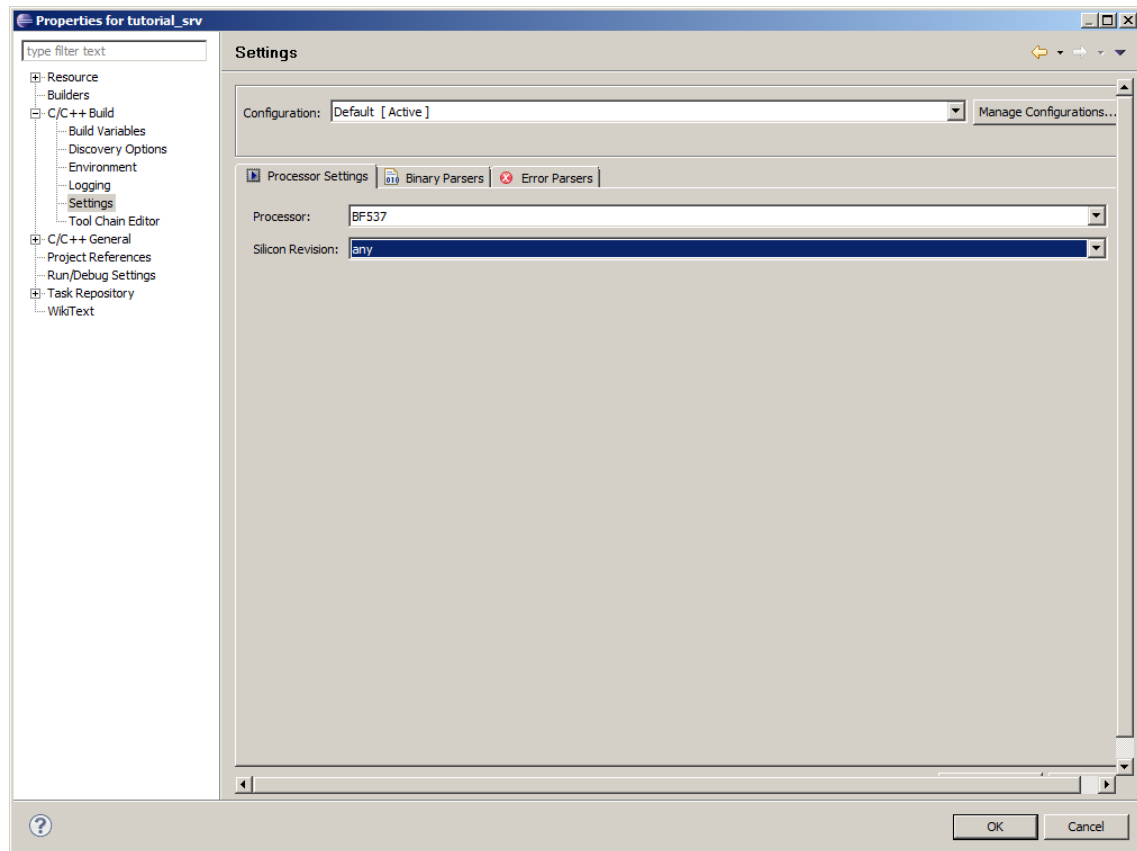


Figura C.10 - Seleção do processador.

6. Nesta fase já se pode passar à compilação. Para isso clicar em “Project” e de seguida em “Build Project”. A compilação será feita e será gerado o ficheiro `srv1.ldr` para transferir para o *flash buffer*. Deverá ser apresentada as mensagens na consola da seguinte figura. Para transferir o ficheiro `srv1.ldr` para o *flash buffer* ver o tutorial *Compilação do firmware e transferência*.

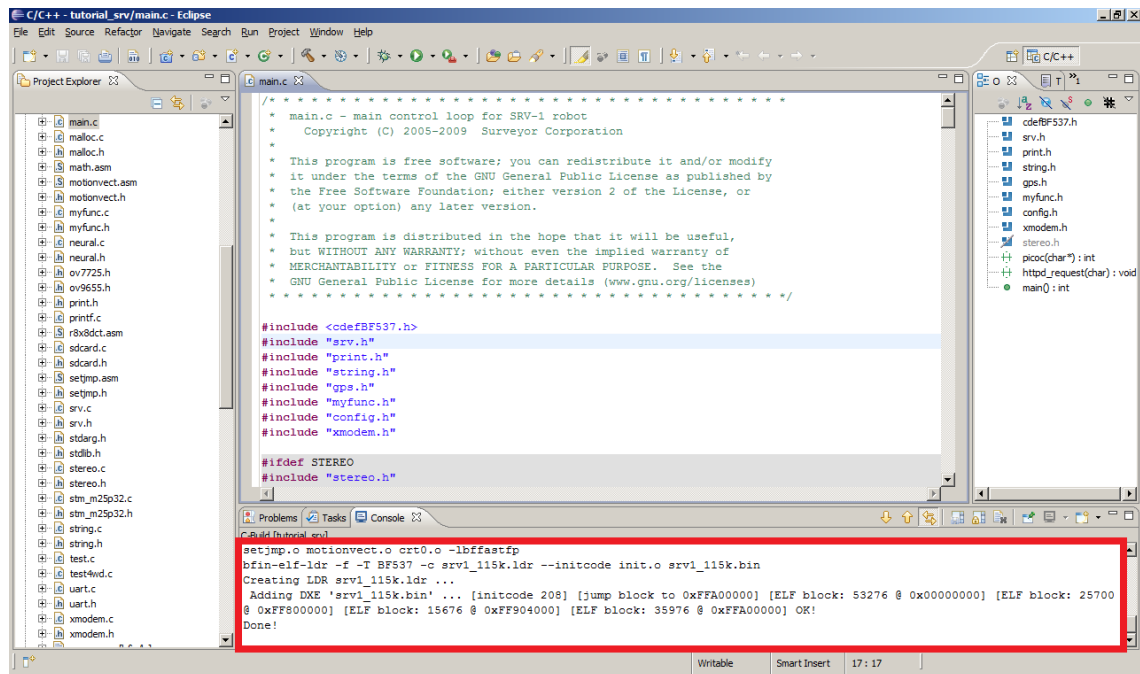


Figura C.11 - Compilação concluída.

# Bibliografia

- [1] A. N. Belbachir, *Smart Cameras*, Viena: Springer, 2010.
- [2] Y. Shi and S. Lichman, "Smart Cameras: Review," Eveleigh, Austrália, 2006.
- [3] Society of Robots, "How Do I Build A Robot?," 2005. [Online]. Available: [http://www.societyofrobots.com/robot\\_faq.shtml#camera\\_microcontroller](http://www.societyofrobots.com/robot_faq.shtml#camera_microcontroller). [Accessed 28 05 2012].
- [4] R. Silva, *Localização de AGVs Industriais Baseada em Marcadores*, Porto, 2011.
- [5] J. M. Cacasús, *Teoria da Imagem*, Rio de Janeiro: Salvat, 1979.
- [6] Rachel Ehrenberg, "Wired Science," 28 06 2010. [Online]. Available: <http://www.wired.com/wiredscience/2010/06/smoothing-square-pixels/>. [Accessed 28 05 2012].
- [7] A. M. R. d. S. F. d. Mendonça, "Introducao\_SBVI\_0910.," FEUP, Porto, 2009.
- [8] A. N. Netravali and B. G. Haskell, *Digital Pictures*, New York: AT&T Bell Laboratories, 1995.
- [9] P. SOMMER and M. ORZESSEK, "ATM & MPEG-2: Integrating Digital Video into Broad," Prentice Hall, 1998.
- [10] M. G. MANZATO, "Técnicas e Padrões de Codificação de Vídeo," Londrina, 2004.
- [11] S. Hetzl, "The .bmp file format," 1998. [Online]. Available: <http://expertsetup.com/~brian/bmp/index.html>. [Accessed 29 05 2012].
- [12] CompuServe Incorporated, "GRAPHICS INTERCHANGE FORMAT," 31 07 1990. [Online]. Available: <http://local.wasp.uwa.edu.au/~pbourke/dataformats/gif/>. [Accessed 29 05 2012].
- [13] J. M. B. Lopes, "Formatos de Imagem," Instituto Superior Técnico, Lisboa, 2008.
- [14] Adobe Developers Association, "TIFF Revision 6.0," Adobe Systems Incorporated, San Jose, 1992.
- [15] C. Steger, M. Ulrich and C. Wiedemann, *Machine Vision Algorithms and*

Applications, München: Wiley-VCH, 2007.

- [16] S. W. Grotta, "Anatomy of a Digital Camera: Image Sensors," 06 12 2001. [Online]. Available: [http://www.smartdevicecentral.com/print\\_article/Anatomy+of+a+Digital+Camera+Image+Sensors/2036.aspx](http://www.smartdevicecentral.com/print_article/Anatomy+of+a+Digital+Camera+Image+Sensors/2036.aspx). [Accessed 30 05 2012].
- [17] J. M. Fonseca, "Sensores de Imagem," Lisboa, 2003.
- [18] R. d. S. Britto, "Uma Arquitetura Distribuída de Hardware e Software para Controle de um Robô Móvel Autônomo," UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE, Natal, Brasil, 2008.
- [19] D. Silva, Í. Batista and T. Varela, "MODELO DE UMA ARQUITETURA DE COMPUTAÇÃO DISTRIBUÍDA," in *IV Congresso de pesquisa e inovação da rede norte de educação tecnológica*, Belém, Brasil, 2008.
- [20] O. M. Filho and H. V. Neto, *Processamento Digital de Imagens*, Rio de Janeiro: Brasport, 1999.
- [21] M. P. d. Albuquerque and M. P. d. Albuquerque, *Processamento de Imagens: Métodos e Análises*, Rio de Janeiro, Brasil: Centro Brasileiro de Pesquisas Físicas, 2005.
- [22] F. F. Feliciano, S. I. Souza and R. F. Leta, "Visão Computacional Aplicada a Metrologia Dimensional Automatizada," Brasil, Engevista, 2005.
- [23] X. A. Falcão and J. N. Leite, "Fundamentos de processamento de imagem digital," 2009.
- [24] T. Jamundá, "Reconhecimento de Formas: A Transformada de Hough," 2005. [Online]. Available: <http://www.inf.ufsc.br/~visao/2000/Hough/>. [Accessed 2012 06 01].
- [25] L. E. B. Dorini and A. d. R. Rocha, "Detecção de círculos em imagens através da transformada de Hough," Brasil, 2004.
- [26] R. O. Duda and E. P. Hart, *Use of the Hough Transformation to Detect Lines and Curves in Pictures*, 1971.
- [27] N. Kiryati, Y. Eldar and A. M. Bruckstein, "A Probabilistic Hough Transform," in *Pattern Recognition*, 1994, pp. Vol. 16, pp. 911-915.
- [28] L. Xu and E. Oja, "Randomized Hough Transform," IGI Global, Pequim, 2009.
- [29] A. Conci and R. H. C. Melo, "Técnicas de Melhorias e Reconhecimento de Formas por Imagens," in *4º Congresso Temático de Dinâmica, Controle e Aplicações*, São Paulo, 2005.
- [30] C. Akinlar and C. Topal, "EDLINES: Real-Time Line Segment Detections By Edge Drawing," in *18th IEEE International Conference on Image Processing*, Brussels,

- 2011.
- [31] C. Topal and C. Akinlar, "Edge Drawing - A Combined Real-Time Edge/Edge Segment Detector," Agosto 2010. [Online]. Available: <http://ceng.anadolu.edu.tr/CV/EdgeDrawing/>. [Accessed 01 06 2012].
  - [32] Surveyor Corporation, "Corporate Background," Surveyor Corporation, 2008. [Online]. Available: <http://www.surveyor.com/>. [Accessed 03 06 2012].
  - [33] Omnivision, *OV775 Color CMOS VGA Sensor*, 2007.
  - [34] Analog devices, *ASDP-BF537 Blankfin Embedded Processor*, 2007.
  - [35] Samsung Semiconductor, *K4S561632J SDRAM Datasheet*, 2007.
  - [36] ST Microelectronics, *M25P32 SPI NAND Flash datasheet*, 2004.
  - [37] Analog Devices, *ADSP-BF53x/BF56x Blackfin® Processor Programming Reference*, USA, 2006.
  - [38] D. d. S. Lima, *Localização Absoluta De Robôs Móveis Em Ambientes Industriais*, Porto, 2010.
  - [39] INESC Porto, "Robvigil," 2011. [Online]. Available: <http://www2.inescporto.pt/rovis/projectos/projectos-em-destaque/robotvigil-1/>. [Accessed 06 06 2012].
  - [40] 5DPO, "5DPO," 2011. [Online]. Available: <http://paginas.fe.up.pt/~robosoc/en/doku.php>. [Accessed 06 06 2012].
  - [41] M. Kohn, "Using SIMD Instructions For Image Processing," 1997-2012. [Online]. Available: [http://www.mikekohn.net/stuff/image\\_processing.php](http://www.mikekohn.net/stuff/image_processing.php). [Accessed 16 06 2012].
  - [42] C. Maas, Christian Maas, 2002. [Online]. Available: <http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm#screenshots>. [Accessed 16 06 2012].
  - [43] Microsoft Corporation, "Converting Between YUV and RGB," 08 04 2004. [Online]. Available: <http://msdn.microsoft.com/en-us/library/ms893078.aspx>. [Accessed 16 06 2012].
  - [44] M. Rigonatto, "Ângulo formado entre duas retas," 2011. [Online]. Available: <http://www.mundoeducacao.com.br/matematica/angulo-formado-entre-duas-retas.htm>. [Accessed 17 06 2012].
  - [45] Surveyor Corporation, "Surveyor SRV-1 Blackfin Camera," 2 7 2010. [Online]. Available: <http://www.surveyor.com/blackfin/>.
  - [46] CmuCam, "Wiki," 2010. [Online]. Available: [http://www.cmuCam.org/projects/cmuCam4/wiki/Frequently\\_Asked\\_Questions](http://www.cmuCam.org/projects/cmuCam4/wiki/Frequently_Asked_Questions). [Accessed 28 05 2012].

- [47] National Instruments, 2012. [Online]. Available: <http://sine.ni.com/nips/cds/view/p/lang/en/nid/204078>. [Accessed 28 05 2012].
- [48] JROBOT, "AVRcam," 2004. [Online]. Available: <http://www.jrobot.net/Projects.html>. [Accessed 28 05 2012].
- [49] PPT Vision, "PPT Vision, A-Series Intelligent Cameras," 2003. [Online]. Available: [http://www.pptvision.com/product\\_a-series.html](http://www.pptvision.com/product_a-series.html). [Accessed 28 05 2012].
- [50] Midsensors, "Vision Subsystem v4 for NXT (NXTCam-v4)," 2005. [Online]. Available: [http://www.midsensors.com/index.php?module=pagemaster&PAGE\\_user\\_op=view\\_page&PAGE\\_id=78](http://www.midsensors.com/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=78). [Accessed 28 05 2012].
- [51] Wikipédia, "YUV," 29 3 2012. [Online]. Available: <http://en.wikipedia.org/wiki/YUV>. [Accessed 03 06 2012].
- [52] Surveyor Corporation, "Surveyor Corporation Store," Surveyor Corporation, 2006. [Online]. Available: <http://surveyor-corporation.stores.yahoo.net/srrowestkit.html>. [Accessed 03 06 2012].
- [53] Recom-Internacional, *R-78xx-1.0 Series Datasheet*, 2007.
- [54] Surveyor Corporation, "Definition of the SRV-1 Control Protocol (Blackfin Version)," 24 Abril 2010. [Online]. Available: [http://www.surveyor.com/SRV\\_protocol.html](http://www.surveyor.com/SRV_protocol.html). [Accessed 2012 Fevereiro 16].
- [55] Surveyor Corporation, "SRV-1 C Interpreter," 7 Maio 2010. [Online]. Available: <http://www.surveyor.com/C.html>. [Accessed 16 Fevereiro 2012].
- [56] Tech, "Vision Light," 2011. [Online]. Available: [www.visionlighttech.com](http://www.visionlighttech.com). [Accessed 2012 06 06].
- [57] H. Kälviäinen, P. Hirvonen, L. Xu and E. Oja, "Probabilistic and non-probabilistic Hough Transforms: overview and comparisons," Elsevier Science, Finlândia, 1995.
- [58] R. Silva, "Localização de AGVs Industriais Baseada em Marcadores," Porto, 2011.