UNIVERSIDADE DA BEIRA INTERIOR
Engenharias

Sistema embutido para detecção, reconhecimento e classificação de sinalética em tempo real

Diogo Veríssimo Correia

Dissertação para obtenção do Grau de Mestre em
Engenharia electrotécnica e de computadores:
Ramo de automação e electrónica
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Miguel F. Dinis O. Gaspar

Covilhã, Outubro de 2010

Agradecimentos

Expresso o meu agradecimento:

- Ao Prof. Dr. Pedro Miguel Dinis Gaspar pelo auxílio prestado na elaboração deste trabalho, assim como ao fornecimento atempado do material requerido, espaço de trabalho e por estar sempre munido com a sua boa disposição e simpatia.
- Aos meus colegas de laboratório de investigação por terem ouvido as minhas ideias em voz alta mesmo não percebendo nada do assunto.
- À Sónia por ter ajudado no apoio psicológico e por ter sido muitas vezes trocada pela dissertação.
- À minha família e amigos em geral, por compreenderem que, por vezes, o trabalho encontra-se em primeiro lugar.

Resumo

Este estudo reside no desenvolvimento de um sistema embutido de reduzida capacidade computacional e baixo consumo energético. O sistema constituído por um processador NXP LPC2106 (NXP, 2008) com arquitectura ARM7, destina-se à obtenção e processamento de imagens, com o intuito de detectar e reconhecer sinais de trânsito portugueses.

Face ao processador utilizado e às características almejadas para o sistema embutido, foi criado um conjunto de algoritmos que requerem reduzidos recursos computacionais e de memória. As principais características deste estudo residem no facto de ter sido maioritariamente construído de raiz permitindo um profundo conhecimento do funcionamento destes algoritmos.

Cada algoritmo foi testado com imagens estáticas, primeiramente utilizando o MATLAB e posteriormente através da programação da CMUcam3. Foi também efectuada a simulação do ambiente de estrada e finalmente foram realizados testes práticos.

As limitações impostas pelo dispositivo levaram ao aumento da complexidade do projecto, porém, o resultado final corresponde a uma taxa de reconhecimento de 81,9%. Neste sentido, pode-se considerar que a taxa de reconhecimento conseguida ultrapassou as expectativas perspectivadas para o sistema embutido.

Palavras-chave

Microcontrolador LPC2106.
CMUcam3.
Reconhecimento de imagem.
Sinais de trânsito.
Processamento em tempo real.
Sistema embutido.

Abstract

This study involves the development of an embedded system with low computational resources and low power consumption. The system constituted by the NXP LPC2106 (NXP, 2008) with an ARM7 processor architecture, acquires and processes images, in order to detect and recognize Portuguese traffic signs.

Considering the processor and the aimed characteristics for the embedded system, a set of algorithms was developed that require low computational resources and memory. The main characteristics of this study consist in the development from scratch which allows a deep knowledge of the algorithms operation.

Each algorithm was tested with static images, first using MATLAB and then by programming the CMUcam3. The road environment was simulated and experimental tests were performed.
The limitations imposed by the device led to increased complexity of the project, however the final result corresponds to recognition rate of 81.9%. Thus, it can be considered that the recognition rate obtained overcome the expectations for this embedded system.

Keywords

LPC2106 microcontroller.
CMUcam3.
Image recognition.
Traffic signs.
Real Time processing.
Embedded system.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API	**A**pplication **P**rogramming **I**nterface.
ARM	**A**dvanced **R**ISC **M**achine.
ART	**A**daptive **R**esonance **T**echnique.
CMOS	**C**omplementary **M**etal–**O**xide–**S**emiconductor.
CPU	**C**entral **P**rocessing **U**nit.
CRT	**C**athode **R**ay **T**ube.
DSP	**D**igital **S**ignal **P**rocessor.
FFT	**F**ast **F**ourier **T**ransform.
FIFO	**F**irst **I**n, **F**irst **O**ut.
FPS	**F**rames **P**er **S**econd.
GCC	**G**NU **C**ompiler **C**ollection.
GPS	**G**lobal **P**ositioning **S**ystem.
GUI	**G**raphic **U**ser **I**nterface.
HSV	**H**ue, **S**aturation, **V**alue.
HSL	**H**ue, **S**aturation, **L**uminance.
I/O	**I**nput/**O**utput.
I^2C	**I**nter-**I**ntegrated **C**ircuit.
JPEG	**J**oint **P**hotographic **E**xperts **G**roup.
LED	**L**ight-**E**mitting **D**iode.
MAM	**M**edia **A**sset **M**anagement.
MMC	**M**ulti**m**edia **C**ard.
NN	**N**eural **N**etwork.
RAM	**R**andom **A**ccess **M**emory.
RBF	**R**adial **B**asis **F**unction.
RGB	**R**ed, **G**reen, **B**lue.
RISC	**R**educed **I**nstruction **S**et **C**omputer.
ROM	**R**ead **O**nly **M**emory.
RS-232	**R**ecommended **S**tandard **232**.
SCCB	**S**erial **C**amera **C**ontrol **B**us.
SPI	**S**erial **P**eripheral **I**nterface bus.
SVM	**S**upport **V**ector **M**achine.
TSDRS	**T**raffic **S**ign **D**etection and **R**ecognition **S**ystem.
USB	**U**niversal **S**erial **B**us.
YCbCr	Luma, **C**hrome **b**lue-difference, **C**hrome **r**ed-difference.

Capítulo 1 - Introdução

Introdução

Como indicado por Bascón *et al*. (2010), a visibilidade dos sinais de trânsito é fundamental para a segurança dos motoristas e pedestres. Por esta razão, os sistemas automáticos desenvolvidos para dar informações sobre sinais de trânsito são uma das questões mais importantes para a investigação de sistemas de transporte inteligentes. Apesar de existirem alguns sistemas baseados na retro-reflectividade tal como o desenvolvido por Siegmann *et al*. (2008), a maioria das pesquisas nesse campo tem sido feita na detecção automática de sinais de tráfegos e sistemas de reconhecimento (*Traffic Sign Detection and Recognition System* - TSDRS) com uma vasta gama de aplicações tais como: condução autónoma, sistemas de assistência à condução e inventário automático de estradas. A maioria dos TSDRS é dividida em três etapas: segmentação, detecção e classificação de pictograma. O papel da primeira fase consiste em isolar regiões candidatas que devem ser classificadas, enquanto o bloco de detecção selecciona as regiões que têm forma de sinal de trânsito. A fase de classificação identifica as informações do sinal de trânsito e/ou determina se a região candidata é apenas ruído. Como referido por Bascón *et al*. (2010) há muitos campos de pesquisa no TSDRS tais como as melhorias na segmentação e blocos de detecção. No entanto, uma das questões mais importantes reside na melhoria da fase de classificação de tal forma que o tempo de reconhecimento possa ser minimizado, mantendo um nível de alta precisão. Geralmente, os sinais de trânsito para esta fase são variáveis sobre as regiões que devem estar devidamente representadas por uma técnica de classificação. Existem duas opções para fazer a classificação:

1- Extrair algumas características da imagem e utilizar esses recursos como entrada para o classificador.

2- Apresentar parte da própria imagem através das intensidades dos pixéis sub-amostrados.

No primeiro caso, cada zona candidata é caracterizada por um vector. Uma solução foi conseguida por Bueno *et al*. (2005) que utilizou métodos de extracção com recurso a histogramas para identificação do sinal de estrada. Outros métodos foram utilizados por Hibi (1996) e Kang *et al*. (1994), fazendo uso de *Fast Fourier Transform* (FFT) calculada após um mapeamento logarítmico complexo dos bordos exteriores, enquanto que Hsu e Huang (2000) e Douville (2000) basearam-se nas formas de onda. A outra abordagem consiste em usar directamente a imagem e nestas condições o tamanho da mesma é considerado. O compromisso entre uma boa resolução para distinguir as classes e capacidade computacional

determina o tamanho para o qual as imagens são dimensionadas. Alguns exemplos de imagens são normalizados em imagens de 16x16 pixéis (Krebel *et al.*, 1999). Maldonado *et al.* (2007) normalizou a imagem a 31x31 pixéis, aplicando também uma máscara antes do processo de reconhecimento, a fim de remover o fundo em relação às informações de forma do sinal. Tendo adoptado as características adequadas, vários métodos de reconhecimento têm sido propostos como Redes Neuronais (NN) por Kang (1994), Aoyagi e Asakura (1996) e de la Escalera *et al.* (1994); rede de Kohonen por Luo *et al.* (1992), onde a rede é treinada considerando rotação e oclusão; rede com teoria da ressonância adaptativa (*Adaptive Resonance Theory* - ART) por de la Escalera *et al.* (2004); e rede com função de base radial (*Radial Basis Function* - RBF) como descrito por Krebel *et al.* (1999). Classificadores alternativos têm sido propostos tal como o classificador de correspondência base por Hibi (1996), um correlator não linear por Perez e Javidi (2002) ou uma combinação de padrões de correlação normalizados baseado numa base de dados usando um sinal de trânsito (Miura *et al.*, 2000; de la Escalera *et al.*, 2004). Segundo Maldonado *et al.* (2007), bons resultados têm sido alcançados quando o método de classificação é baseado em máquinas de suporte vectorial (*Support Vector Machine* - SVM) com filtro Gaussiano como demonstrado por Frias-Martinez *et al.* (2006), Krebel *et al.* (1999) e Cyganek (2007), no entanto, a identificação do sinal num TSDRS é tecnicamente difícil de realizar devido aos seguintes problemas:

1- As condições de iluminação do cenário são mutáveis de acordo com a hora do dia ou da noite e afectam a percepção da cor de sinais de trânsito. Assim, o nível de iluminação tem uma grande influência sobre os vectores de identificação.

    a. Para o mesmo cenário, o uso de câmaras diferentes e configurações diferentes para a mesma câmara podem provocar alterações na imagem.

    b. Embora os sinais de trânsito sejam objectos feitos pelo homem definidos por normas internacionais, existem pequenas variações em relação aos pictogramas de acordo com os fabricantes.

2- A variedade de sinais de trânsito a ser distinguido é enorme em cada país.

Para além destes sistemas de cognição, bem como lidar com todas estas questões, são também de evitar a identificação errónea de falsos sinais, ou seja, limitar o número dos falsos alarmes e, finalmente, mesmo quando o objectivo não é orientado para aplicações em tempo real, um elevado tempo de computação. Para além disso, o sistema necessita ser avaliado com uma base de dados de tráfego de sinais reais contendo um número suficientemente grande de exemplos.

Um exemplo de TSDRS foi realizado por Shojania (2003), que utilizou o método desenvolvido por de la Escalera *et al.* (2003) e construiu um algoritmo capaz de detectar sinais de trânsito da seguinte forma:

1- Segmentação de cor;
2- Detecção de cantos;
3- Reconhecimento da forma;
4- Classificação de sinais.

Com este método, Shojania (2003) pôde fazer o reconhecimento de sinais de trânsito mesmo com informação em falta (com o sinal parcialmente obstruido, por exemplo, por uma árvore), o que é algo que pode acontecer num cenário real. Contudo, também caracterizava erroneamente reconhecendo como objectos elementos de cor vermelha, como por exemplo, veículos ou mesmo a roupa de pessoas.

Actualmente, a investigação tem levado à capacidade de detectar e reconhecer também padrões no geral, mais especificamente como a detecção de rostos através da utilização do algoritmo desenvolvido por Viola e Jones (2001) que decompõe o rosto humano em caracteristicas fazendo com que seja de baixo recurso computacional.

1.1. Objectivos e estratégias

O objectivo desta dissertação é, através da utilização de um processador de sinal, obter e reconhecer sinais de trânsito portugueses.
Para tal, o problema será abordado da seguinte forma:

1.1. Utilizar o sistema de cores formadas pelas componentes Matiz (*Hue*), Saturação (*Saturation*) e Valor (*Value*) (*Hue, Saturation, Value* - HSV) em detrimento do sistema de cores aditivas formado por Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*) (*Red, Green, Blue* - RGB) para filtrar de forma mais eficiente os sinais de trânsito, recorrendo-se para tal a uma conversão computacional.

1.2. Adicionar algoritmos à abordagem de Paulo (2000), Carlson *et al.*(2008) e Shojania (2003) para reduzir os erros de reconhecimento de imagem nomeadamente na detecção de falsos positivos.

1.3. Implementar todo este processo no sistema embutido da NXP (NXP, 2008), considerando um limitado espaço de armazenamento e de processamento.

1.2. Organização do relatório

Este relatório encontra-se dividido em 7 capítulos:

No 1º capítulo é feita a apresentação do trabalho, composta pelo estado da arte, e objectivos e estratégias delineados para a conclusão da dissertação;

O 2º capítulo contém a descrição dos diferentes sistemas/métodos de codificações de cores.

No 3º capítulo encontra-se descrito o *hardware* e *software* utilizado para o desenvolvimento do sistema embutido de detecção e reconhecimento de sinaléticas em tempo real;

O 4º capítulo é composto pela descrição detalhada dos procedimentos inerentes aos trabalhos conducentes à obtenção dos objectivos delineados;

O capítulo 5 apresenta excertos de linhas de código e respectivos comentários, relativos ao programa que contém os algoritmos responsáveis pela detecção e reconhecimento de sinaléticas;

No capítulo 6 são descritos os testes experimentais realizados e é efectuada uma análise aos resultados obtidos.

O capítulo 7 encerra a dissertação com as conclusões do trabalho e perspectivas de trabalho futuro.

Capítulo 2 - Sistemas de codificação de cores

Introdução

Conforme indicado por Boughen (2003), existem diferentes sistemas de codificação de cores nos dispositivos de imagem, sendo o mais comum o sistema RGB (*Red, Green, Blue*) que, com as três componentes de cor consegue produzir uma imagem a cores. Esta codificação de cores é composta por uma matriz de 3 dimensões (largura x altura x cores), em que as duas primeiras (largura x altura) dão o valor de pixéis. Trata-se do formato comummente utilizado por se poder fazer facilmente a separação de cada componente e posterior emissão, quer seja em tubos de raios catódicos quer através da iluminação de um diodo emissor de luz, vulgarmente denominado LED (*Light-Emitting Diode*) (da cor correspondente). A imagem seguinte é um exemplo gráfico da formação das cores.

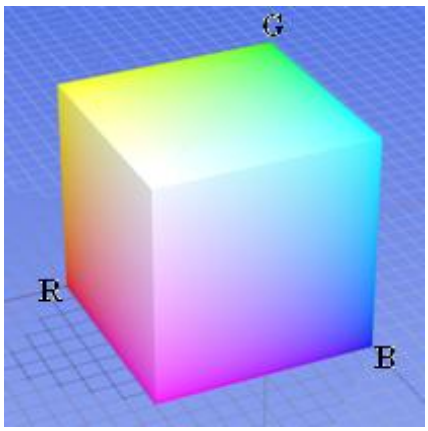


Figura 1 - Modelo de cores RGB mapeado para um cubo.

YCbCr (*Luma, Chrome blue-difference, Chrome red-difference*) é um sistema de codificação de cores usado em imagem de vídeo e sistemas de fotografia digital. Não se trata de um espaço de cor absoluto, mas sim uma forma de codificar informação RGB como descrito por Boughen (2003). A cor real exibida depende do RGB usado para exibir o sinal. Portanto, um valor expresso em YCbCr só é previsível se for usado o padrão do RGB ou um perfil ICC (*International Color Consortium*). A componente Y (*Luma*) é a informação a preto e branco da imagem (que ocupa a maior largura de banda na transmissão), as componentes Cb e Cr (*Chrome blue-difference, Chrome red-difference)* são imagens subtractivas da cor RGB com a componente Y e que podem ser transmitidas sub-amostradas de modo a reduzir a largura de banda total de transmissão. Desde modo poder-se-á visualizar uma imagem com apenas a componente Y, obtendo uma imagem monocromática.

Estes dois sistemas de codificação de cores estão fortemente relacionados com a nossa percepção das cores do mundo real, juntando desta forma dois espaços de cor, o RGB e a luminância e crominância. Estes permitem efectuar a ligação entre a necessidade de *hardware* com a necessidade de transmissão.

Um outro espaço de cores utilizado é o HSV (*Hue*, *Saturation*, *Value*) que provem do cálculo (apresentado no sub-capítulo seguinte) da cor RGB e que segundo Paulo (2000) tem como principal contributo a forma de como é codificada a cor e efectuada a separação das suas componentes. Segundo Boughen (2003) a componente matiz (*hue*) provem da cor pura. Esta cor encontra-se no espectro visível quer seja primária ou mistura de cores e é comparada com o que é visível no arco-iris; a componente saturação (*saturation*) consiste na quantificação da cor, por outras palavras, reside na pureza da cor. Um exemplo pode ser dado relativamente à distinção entre um vermelho saturado que significa um vermelho primário e um rosa que é considerado vermelho menos saturado; finalmente a componente valor (*value*) agrega a cor entre a cor mais densa (quando o valor é elevado) e o preto (quando o valor é reduzido). A imagem seguinte é um exemplo gráfico da formação das cores.

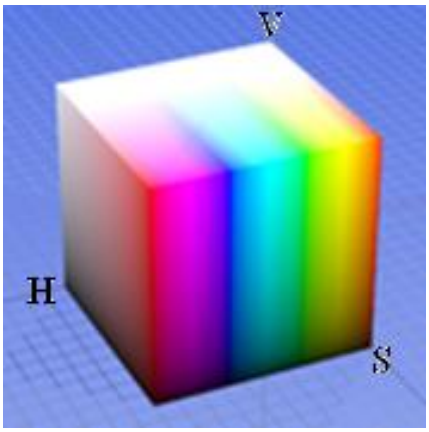


Figura 2 - Modelo de cores HSV mapeado para um cubo.

O espaço de cores HSL (*Hue*, *Saturation*, *Luminance*) é semelhante ao espaço de cores HSV, uma vez que são ambas cores de coordenadas circulares. Contudo o HSL é uma codificação que exige conhecer o valor mais elevado e mais baixo de componentes de RGB da imagem, o que significaria um aumento significativo, ou mesmo não realizável, das necessidades computacionais por parte do microcontrolador.

2.1. Utilização dos sistemas de codificação RGB, YCbCr, HSV e HSL

Os tubos de raios catódicos (televisores CRT - *Cathode Ray Tube*) são impulsionados por sinais de tensão pelo vermelho, verde e azul, mas estes sinais RGB não são eficientes como uma representação para o armazenamento e transmissão, uma vez que têm muita redundância.

O sistema YCbCr é uma aproximação prática à transformação de cores e uniformidade de percepção, onde as cores primárias que corresponde aproximadamente a vermelho, verde e azul são transformadas em informação percentualmente significativa. O sistema YCbCr é usado para separar um sinal de luma (*Y*) que pode ser armazenado com alta resolução ou transmitido em banda larga, e dois componentes cromáticos (*Cb* e *Cr*) que podem ser de largura de banda reduzida, sub-amostrados, comprimidos, ou de outra forma tratados separadamente para melhorar a eficiência do sistema conforme demonstrado por Boughen (2003).

Desta forma a utilização dos sistemas RGB e YCbCr está sempre presente na codificação de imagem.

Segundo Boughen (2003) os sinais YCbCr são criados a partir do RGB de origem usando duas constantes definidas *Kb* e *Kr* como segue:

$$Y = Kr \times R + (1 - Kr - Kb) \times G + Kb \times B \tag{1}$$

$$Cb = \frac{1}{2} \times \frac{B - Y}{1 - Kb} \tag{2}$$

$$Cr = \frac{1}{2} \times \frac{R - Y}{1 - Kr} \tag{3}$$

Onde:

Y = componente luma (informação);

Cb = Componente cromática de diferença de azul;

Cr = Componente cromática de diferença de vermelho;

R = componente vermelho do sinal de entrada (RGB);

G = componente verde do sinal de entrada (RGB);

B = componente azul do sinal de entrada (RGB);

Kr = variável aplicada à componente vermelha;

Kb = variável aplicada à componente azul;

Nota: As variáveis Kr e Kb são definidas pela ICC para os diferentes dispositivos.

A utilização de um espaço de cores RGB ou YCbCr tornará difícil o reconhecimento de sinais com variações de iluminação, pelo que uma abordagem adequada reside na utilização do espaço de cores HSV, uma vez que a matiz não é influenciada pela luz solar (luminância). Para tal, a Matiz, a Saturação e o Valor terão de ser calculados por *software* segundo as seguintes expressões descritas por Boughen (2003);

$$H_{R,G,B} = \begin{cases} 0, & \max = \min \\ 60^{o} \times \dfrac{G-B}{\max - \min}, & \max = R \cap G \geq B \\ 60^{o} \times \dfrac{G-B}{\max - \min} + 360^{o}, & \max = R \cap G < B \\ 60^{o} \times \dfrac{B-R}{\max - \min} + 120^{o}, & \max = G \\ 60^{o} \times \dfrac{R-G}{\max - \min} + 240^{o}, & \max = B \end{cases} \quad (4)$$

$$S_{\max} = \begin{cases} 0, & \max = 0 \\ 1 - \dfrac{\min}{\max}, & c.c. \end{cases} \quad (5)$$

$$V = \max \quad (6)$$

Onde:

H = hue = matiz;

S = saturation = saturação;

V = value = valor;

R = red = vermelho;

G = green = verde;

B = blue = azul;

max = valor máximo encontrado;

min = valor mínimo encontrado.

Conforme indicado por Paulo (2000), Carlson *et al*. (2008) e Shojania (2003), a utilização de uma imagem em formato HSV tem como principal vantagem a abstracção da luz que poderá ser mais ou menos intensa. Apenas fazendo uma filtragem das componentes H e S se pode seleccionar uma cor. Esta condição é relevante caso seja utilizada uma binarização de cores. A binarização é descrita ao longo deste documento com o intuito de salientar os fundamentos da sua utilização.

Contudo, existem inconvenientes, entre os quais, a necessidade de processamento. Este factor deve ser levado em consideração aquando da utilização deste método de transformação de imagem de RGB para HSV, visto que o sensor CMOS (*Complementary Metal-Oxide-Semiconductor*) não tem a capacidade de transmitir imagens em HSV, nem mesmo o microcontrolador consegue realizar a operação por *hardware*. Este e outros factores levarão a que o processo seja mais demorado. Um exemplo indicativo do tempo dispendido nesta operação será apresentado no capítulo seguinte. Por motivos relacionados com a necessidade de memória e velocidade de transferência, a conversão HSL é descartada.

2.2. Nota conclusiva

Neste capítulo são explorados dois espaços de cores principais (RGB e HSV), sendo demonstrados os motivos que levarão à utilização do espaço de cores HSV e as impossibilidades de se utilizar o espaço de cores HSL.

Em suma, o espaço de cores HSV tem a capacidade de poder seleccionar as cores necessárias para a detecção e reconhecimento dos sinais de trânsito com apenas duas componentes (H e S) e tem a complementaridade de se abstrair de variações do valor da cor (V), isto porque os sinais de trânsito podem variar este parâmetro com as variações de iluminação dos mesmos.

Capitulo 3 - *Hardware* e *software*

Introdução

Este capítulo aborda as especificações do *hardware* e *software* utilizados nesta dissertação. São evidenciadas as potencialidades e factores relacionados com o *hardware*, tais como consumo energético, tempos de execução, módulos de dispositivos, entre outros, que se reflectem na qualidade do sistema embutido desenvolvido. Relativamente ao *software*, é realizada a comparação com outros programas, nomeadamente no que toca à disponibilização de livrarias.

De forma a auxiliar a exemplificação dos conteúdos, faz-se o recurso a figuras que elucidam sobre as características do circuito integrado NXP LPC2106 e do módulo CMOS. Para o *software*, são apresentados, com recurso a tabelas e gráficos de barras, os tempos de execução de algumas instruções.

3.1. *Hardware*

O *hardware* utilizado consiste na CMUcam3, cuja arquitectura é composta por três componentes principais: uma câmara CMOS, um *buffer* de imagem, e um processador digital de sinal (*Digital Signal Processor* - DSP). Segundo Rowe *et al*. (2007), o DSP configura o sensor CMOS, usando um protocolo série de dois fios (SCCB). Em seguida, este inicia uma transferência de imagens directamente da câmara CMOS para o *buffer*, o qual deve aguardar o início de uma nova trama a ser sinalizada, altura em que configura o sistema de forma assíncrona para guardar a imagem no *buffer*. O fim da imagem provoca uma interrupção de *hardware*.

Conforme apresentado na figura seguinte, a CMUcam3 é composta por duas portas séries, com protocolos de comunicação I^2C (*Inter-Integrated Circuit*) e SPI (*Serial Peripheral Interface bus*), quatro saídas *servo* standard, três LED, um botão de pressão e um conector MMC (*MultiMedia Card*). Um cenário típico de funcionamento consiste na comunicação entre um microcontrolador e a CMUcam3 através de uma ligação série (RS-232). Alternativamente, a comunicação pode ser realizada por I^2C ou SPI, tornando a CMUcam3 compatível com a maioria dos sistemas incorporados, sem depender exclusivamente da ligação RS-232.

O barramento SPI é também usado para comunicar com armazenamento em *flash* ligado ao conector MMC permitindo a leitura e escrita de megabytes de memória não volátil.

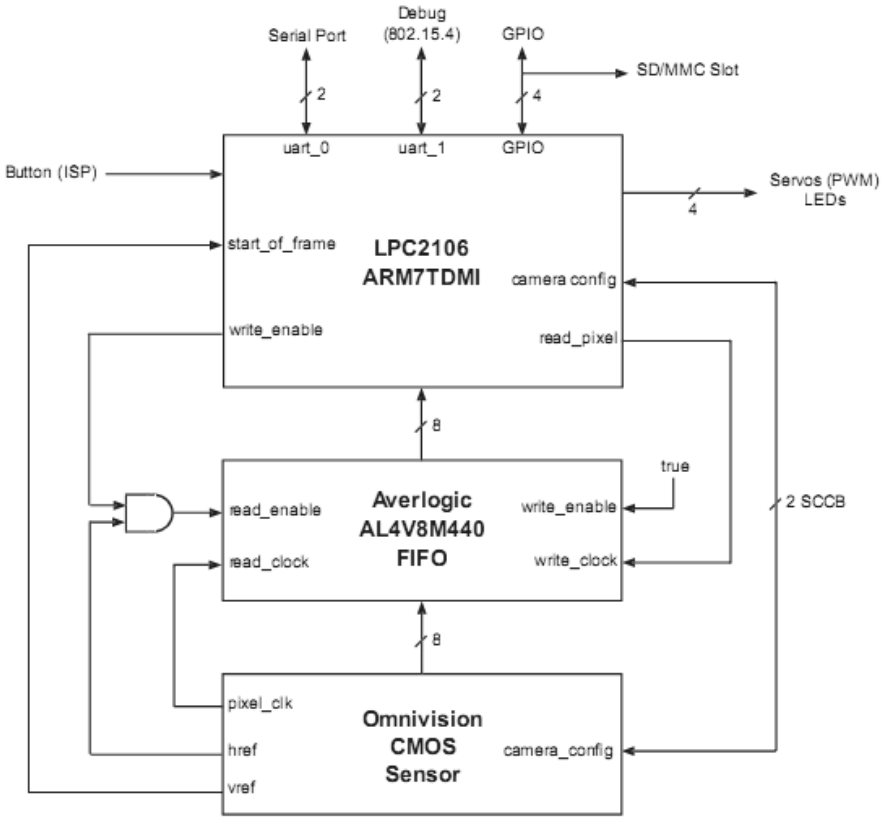


Figura 3 - Esquema da CMUcam3. Fonte: cmucam.org (2009)

Todos estes periféricos são controlados pelo *hardware* e, portanto, não invalidam o tempo de processamento. A porta de expansão na CMUcam3 é compatível com várias redes de sensores sem fios, incluindo o Telos (Polastre *et al.*, 2005). A entrada da imagem do sistema é fornecida por uma câmara OV6620 da Omnivision (Omnivision, 2000) cujo diagrama esquemático se encontra apresentado na figura seguinte. Esta encontra-se montada numa placa que inclui uma lente e componentes passivos. A placa da câmara tem como saída um fluxo de 8-bit RGB ou pixéis de cores YCbCr. O módulo OV6620 suporta uma resolução máxima de 352 × 288 pixéis a 50 imagens por segundo. Parâmetros da câmara, como saturação de cor, brilho, contraste, os ganhos de balanço de branco, tempo de exposição e os modos de produção são controlados usando o protocolo SCCB (*Serial Camera Control Bus*) de dois fios. A sincronização de sinais, incluindo um relógio de pixel (directamente ligado à imagem FIFO - *First In, First Out*), é efectuada para ler os dados e indicar novas imagens, bem como linhas horizontais. A câmara também fornece um sinal monocromático analógico, tal como referido por Rowe *et al.* (2006).

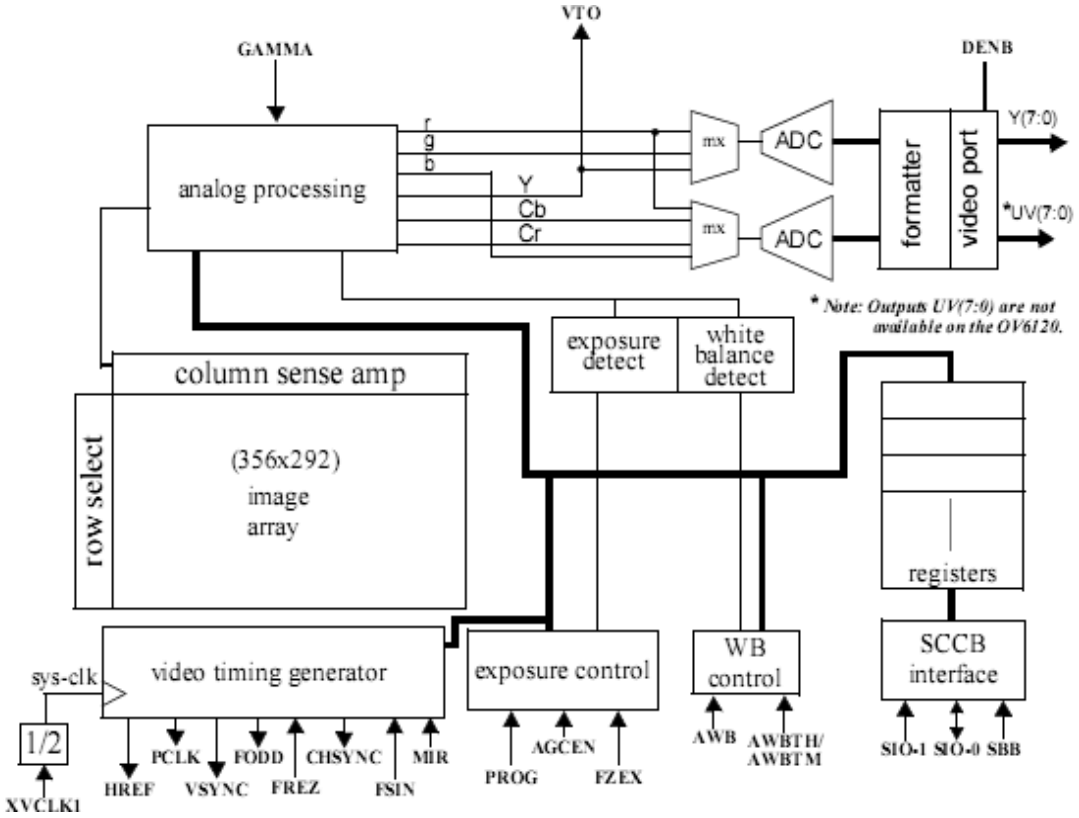
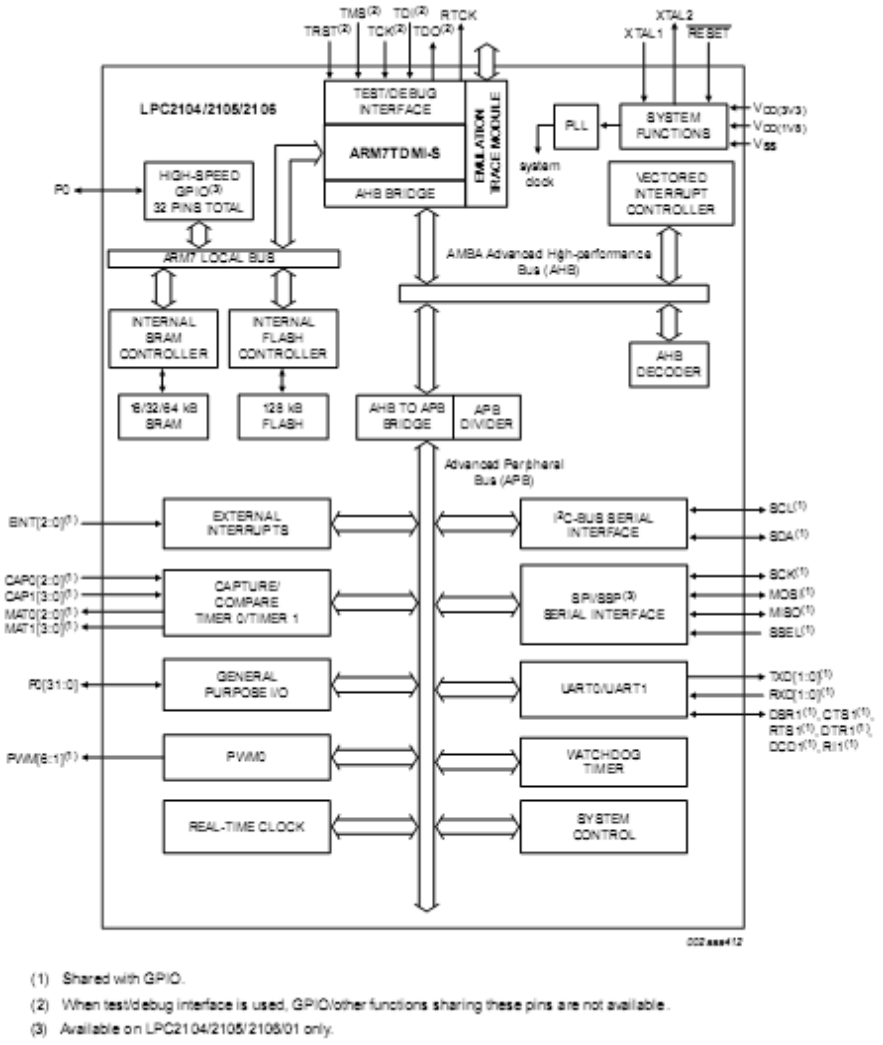


Figura 4 – Estrutura interna do módulo Omnivision OV6620. Fonte: Omnivision

Na raiz do dispositivo está o DSP da NXP, o ISA LPC2106 com 32-bit a ≈15 MHz numa arquitectura ARM7TDMI com 64 kb de RAM e 128 kb de memória *flash* (NXP, 2008). O diagrama esquemático do DSP é apresentado na figura seguinte. O processador é capaz de, por *software*, controlar a escala de frequência e tem uma aceleração de módulo de memória (MAM - *Media Asset Management*), que prevê a colecta de dados da memória *flash* no próximo ciclo. Um carregador embutido *boot* permite fazer o *download* de um arquivo executável através da porta série sem *hardware* de programação externa. Uma vez que o processador utiliza o conjunto de instruções ARM, o código pode ser compilado com o compilador GCC disponível gratuitamente na GNU (CodeSourcery, 2009). Sendo o hardware embutido, o custo é reduzido. Como o suporte do compilador é livre, o LPC2106 torna-se num processador ideal para o desenvolvimento de código aberto (Cygwin, 2010).



(1) Shared with GPIO.
(2) When test/debug interface is used, GPIO/other functions sharing these pins are not available.
(3) Available on LPC2104/2105/2106/01 only.

Figura 5 - Esquema interno do DSP LPC2106 da NXP. Fonte: NXP

O *buffer* de imagens na CMUcam3 a 50MHz é o 1 MB AL4V8M440 FIFO de vídeo produzido pela Averlogic (Rowe *et al.*, 2006). O FIFO de vídeo é importante porque permite que a câmara funcione em alta velocidade e desacople o processamento da CPU (*Central Processing Unit*) da câmara. A câmara, com rendimentos de imagem inteira melhora a taxa de ganho automático e desempenho do sensor CMOS. Apesar dos pixéis não poderem ser acedidos de forma aleatória, o FIFO não permite redefinir o ponteiro de leitura que permite processamento de imagens múltiplas. Como descrito por Rowe *et al.* (2006), uma desvantagem do LPC2106 reside nos portos de entrada/saída (*Input/Output* - I/O) relativamente lentos o que faz com que a leitura de um valor de pixel possa levar 14 ciclos de relógio, dos quais 12 são de espera pela actualização do estado do I/O. Dado que a transferência é desacoplada em cada pixel em tempos de acesso, o relógio de pixel da câmara não precisa ser definido para o tempo de processamento de pixel. Por sua vez, permite o aumento das taxas de imagens, o que não seria possível sem o *buffer*.

Em aplicações autónomas, o consumo de energia é um factor importante. Para facilitar a economia de energia, são fornecidos três modos de operação (activo, inactivo e desligado), bem como a capacidade para desligar apenas o módulo da câmara. No modo de operação activo, o sistema consome 500 mW quando a CPU, câmara e FIFO estão em pleno funcionamento. A tabela seguinte mostra a distribuição do consumo de energia para os diversos componentes. Quando em estado inactivo, com manutenção da RAM (*Random Access Memory*) e câmara inactiva, o sistema consome cerca de 300 mW. O tempo de transição entre inactivo e activo é da ordem de 30 µs.

Tabela 1 - Tabela de consumo dos diferentes dispositivos da CMUcam3. Fonte: cmucam.org

	Tensão (V)	Corrente (mA)	Potência (mW)
Núcleo CPU	1,8	60	108,0
Periféricos CPU	3,3	15	49,5
Buffer imagem	3,3	52	171,0
Câmara	5,0	25	125,0
MMC	3,3	4	13,2
MISC	3,3	10	33,0
Total	-	-	499,7

Para aplicações onde são necessários ciclos temporais muito longos e os atrasos de inicialização de até 1 segundo possam ser tolerados, activando um pino externo, o consumo é quase nulo (25 µW). No entanto, neste estado de funcionamento, a memória RAM do processador não é mantida e, portanto, os parâmetros da câmara devem ser restaurado pelo *firmware* na inicialização.

3.2. *Software*

Sistemas como o OpenCV fornecido pela Intel (2001), LTI-Lib descrito por Kraiss (2004) e MATLAB produzido pela MathWorks (2008) exigem megabytes de espaço de endereçamento de memória e são escritos em linguagens de programação pesadas, como o C++ e Java. A CMUcam3 tem apenas 64 kb de memória RAM e, portanto, não pode usar qualquer uma dessas bibliotecas de visão padrão.

Para resolver esse problema, é concebido e implementado o sistema de visão CC3 (CMUcam3) como o *software* principal para CMUcam3. Também são implementados vários componentes juntamente com o CC3.

O sistema CC3 é uma API C (*Application Programming Interface* C) para a realização da visão e controlo, optimizados para ambientes pequenos como o da CMUcam3.

Segundo Rowe *et al*. (2006), as suas características principais são:

- Camada de abstracção para interface com sistemas de *hardware* futuro.
- Estilo C99 que segundo *Leaps* é um estilo moderno com nomes tipos e funções consistentes.
- Suporte de um número limitado de formatos de imagem de simplicidade.
- Documentação fornecida através do Doxygen, como exemplificada por van Dimitri (2007).
- API preparada para futura extensibilidade.
- Módulo *virtual-cam* para PC para teste e depuração.

O módulo *virtual-cam*, parte do sistema CC3, fornece um ambiente simulado para testar a biblioteca e código do projecto em qualquer computador padrão, através da compilação com o compilador GCC nativo. Isto permite a utilização plena das ferramentas de depuração para diagnosticar problemas no código do utilizador, permitindo analisar erros de programação quando os mesmos surgem devido a uma referência errada ou a um ponteiro de memória inválido. Embora nem todas as funcionalidades CMUcam3 sejam implementados na *virtual-cam* (falta incluir os recursos de *hardware*, componentes específicos de controlo, servo e I/O), possui as funcionalidades suficientes para permitir testes de diagnóstico *off-line*.

Dependendo da resolução da imagem e da complexidade do algoritmo, os tempos de execução podem variar significativamente. O objectivo é fornecer alguma intuição para os diversos tipos de processamento de imagens que são possíveis com a CMUcam3 e entender onde são encontrados "estrangulamentos" no I/O.

A tabela seguinte mostra a repartição do tempo consumido pelos três principais passos envolvidos no carregamento de uma imagem na memória do processador. Os valores "Carregar imagem" referem-se ao tempo necessário cada vez que uma nova imagem chega e

antes que os dados possam começar a ser recuperados do *buffer* de imagem. Os valores "Copiar para memória" referem-se ao tempo necessário para mover os dados do *buffer* de imagem para o processador. Operando em uma resolução menor, obviamente, o tempo de execução é diminuído já que um número inferior de pixéis é acedido. Operando num único canal, em vez de três canais, oferece uma redução de apenas 62,5% no tempo. Esta redução deve-se a não ter que ler todos os pixéis de cor. No entanto, uma vez que a câmara CMOS não tem um modo de produção monocromática, a informação da cor deve ainda ser cronometrada para fora do FIFO. Finalmente a secção "Comprimir pixel" mostra o tempo necessário para converter o padrão GRGB (filtro mosaico de Bayer) da câmara para a estrutura local do pixel RGB na memória.

Tabela 2 - Tempo necessário para imagem CIF e QCIF quer em RGB, quer em monocromático. Fonte: cmu.org

	CIF RGB	CIF Mono	QCIF RGB	QCIF Mono
Carregar imagem (ms)	2	2	2	2
Copiar para memória (ms)	210	128	52	32
Comprimir pixéis (ms)	150	160	38	39
Total FPS	2,76	3,45	10,87	13,7

É possível reduzir significativamente o tempo de conversão projectando algoritmos que operam sobre a memória da câmara. Isto torna-se um compromisso entre o código portátil simples e velocidade de execução.

A figura seguinte mostra o tempo de execução em relação ao consumo das operações de carregamento já mencionadas, juntamente com os tempos de processamento em três diferentes algoritmos: JPEG (*Joint Photographic Experts Group*), seguimento de cor RGB e seguimento de cor YCbCr. Neste exemplo, o algoritmo JPEG comprime uma imagem colorida em memória e não a escreve para um dispositivo de armazenamento. O *Track Color* (seguimento de cor) [TC] e *Track Color HSV* (seguimento de cor em HSV) [TC-HSV] são algoritmos obtidos directamente do código de emulação anterior (CMUcam2).

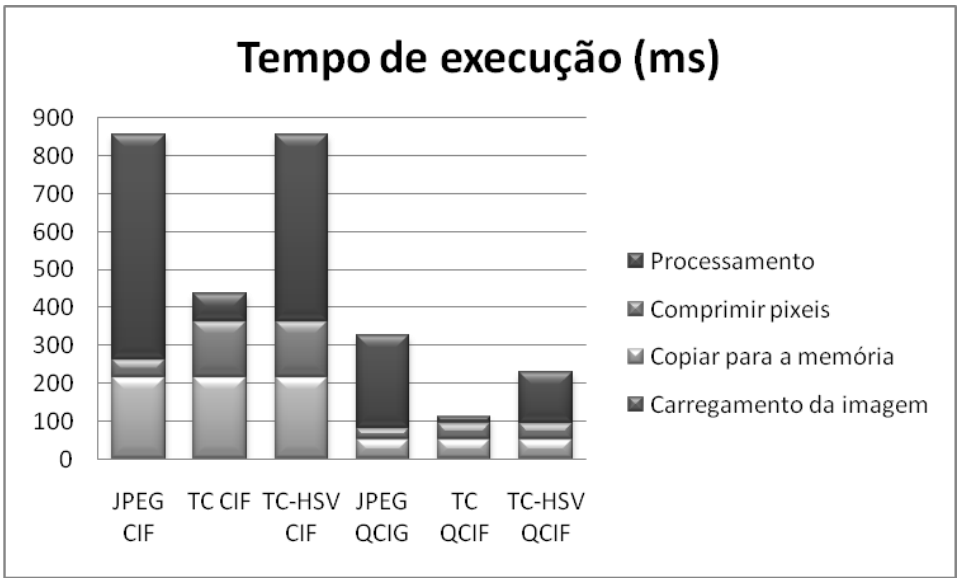


Figura 6 - Consumo de tempo para realização de operações de imagem.

Cada algoritmo encontra a caixa delimitadora, centro e densidade de uma determinada cor especificada. Para este teste é mostrado o desempenho do pior caso através do rastreamento de todos os pixéis activos. A referência de cor HSV é idêntica à RGB, excepto no que toca à execução de uma conversão baseada em *software* do RGB para HSV para cada pixel. A tendência geral encontrada nestes algoritmos muito simples, tais como o rastreamento de cor, reside na sua maioria apresentar um I/O limitado. Por exemplo, o seguimento de cor consome apenas 17% do tempo de processamento. Um algoritmo mais complexo, JPEG, consome 62% do tempo de processamento. Como mencionado anteriormente, o LPC2106 possui 64 kb de RAM interna e 128 kb de ROM (*Read Only Memory*). Por defeito, 9 kb de RAM é reservado para espaço de pilha e 9 kb de memória RAM é usada pelas bibliotecas do núcleo (incluindo os *buffers* libc). A 176 × 144 (QCIF) cinza requer 25 kb de memória RAM, enquanto uma imagem de 100 × 100 RGB requer 30 kb de memória. Todo o processamento de imagens em tamanho maior deve ser executado uma a seguir à outra, ou usando uma janela deslizante de digitalização com abordagem de linha. Por exemplo, um JPEG requer apenas oito linhas completas (8 kb) da imagem, além do armazenamento necessário para a imagem compactada (12 kb ou menos). O espaço de códigos consumidos pela maioria das aplicações CMUcam3 é muito pequeno. Um programa simples que carrega imagens e ligações nas funções da biblioteca padrão requer 52 kb de ROM. O sistema de arquivos FAT e controlador MMC exige um adicional de 12 kb de memória ROM.

3.3. Nota conclusiva

Desta secção deve-se sedimentar que a utilização de um *buffer* de imagem tem os benefícios de desacoplamento entre controlador e sensor CMOS fazendo com que ambos possam funcionar a velocidades diferentes. Também se deve ter em consideração que se trata de um equipamento de baixo consumo e de baixo custo, com programador embutido e *software* livre, o que por um lado o torna num investimento reduzido, mas por outro, é de iniciação e desenvolvimento mais difícil.

Capítulo 4 - Fases do processo de detecção e reconhecimento de sinalética

Introdução

Neste capítulo são apresentados os processos inerentes à detecção e reconhecimento de sinaléticas em tempo real. Encontram-se descritas as linhas gerais da actuação fazendo também uma pequena especificação de cada um dos factores.

A resolução do problema é edificada na forma de fluxogramas que fazem referência aos pontos de aquisição, processamento, classificação e comunicação.

4.1. Geral

O processo de detecção e reconhecimento de sinaléticas é composto por diferentes fases. Estes patamares permitem aferir a evolução dos trabalhos e o seu tempo de execução. De salientar que as fases que fazem parte do processo de detecção e reconhecimento de sinaléticas requerem a finalização da fase anterior. De uma forma geral, o trabalho é faseado pelos processos indicados na figura seguinte.

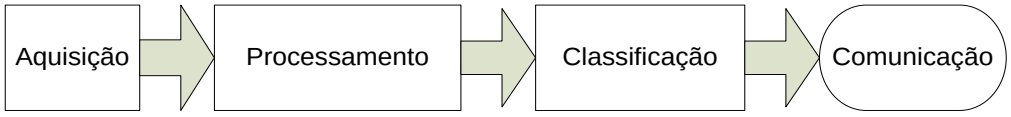


Figura 7 - Fases do processo de detecção e reconhecimento de sinaléticas

Os seguintes sub-capítulos apresentam de forma detalhada cada um dos processos que fazem parte do faseamento dos trabalhos na óptica do desenvolvimento de um sistema embutido destinado à detecção e reconhecimento de sinaléticas em tempo real.

4.2. Aquisição

Neste bloco é realizada a conversão de GRGB (imagem crua) para RGB ou YCbCr. Em termos de *hardware* é caracterizado pela câmara e *buffer* de imagem. A saída do bloco de aquisição consiste na imagem armazenada no *buffer* pelos motivos enunciados no capítulo 2.

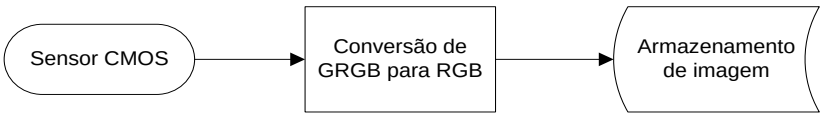


Figura 8 - Descrição do bloco de aquisição

4.3. Processamento

Por processamento considera-se a fase onde é efectuada a extracção dos sinais. Para tal, o procedimento é constituído pelas seguintes operações:

- Converter imagem de RGB para HSV.
- Filtrar a imagem do processo anterior nas diferentes componentes.

Embora seja um processo com processamento relativamente reduzido, é de salientar a importância de uma boa filtragem de dados (pixéis), fazendo desta forma com que os processos seguintes sejam mais rápidos e que se obtenham bons resultados de reconhecimento e classificação de sinais de trânsito.

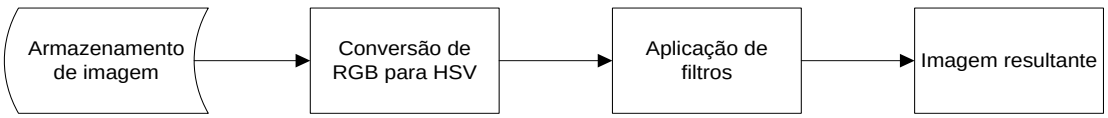


Figura 9 - Descrição do bloco de processamento.

4.4. Classificação

O bloco de classificação é aquele onde nesta dissertação foi desenvolvido mais trabalho inovador. O bloco de classificação recebe um conjunto de imagens em formato binarizado e aplica, numa primeira fase, filtros de detecção de forma. De seguida e caso hajam pictogramas, cria duas imagens de amostragem (dimensão 20 x 20) nas cores branca e preta e posteriormente compara com sinais conhecidos. Se não for detectado um sinal de trânsito, o pictograma é procurado numa região de sinais menos comuns, demorando assim mais tempo. Numa fase seguinte, caso não seja reconhecido o pictograma, a imagem é descartada fazendo com que possa dar oportunidade à próxima imagem, que ao ter outro ângulo ou luminosidade, poderá permitir o seu reconhecimento com sucesso. Um dos parâmetros levado em conta nesta investigação reside num falso positivo ser tão ou mais perigoso que o não

reconhecimento. Neste sentido, o objectivo fundamental deste sistema embutido concentra-se em ter a "certeza" que a identificação do sinal de trânsito reconhecido é correcta. Na figura seguinte encontra-se o fluxograma esquemático do processo de classificação.

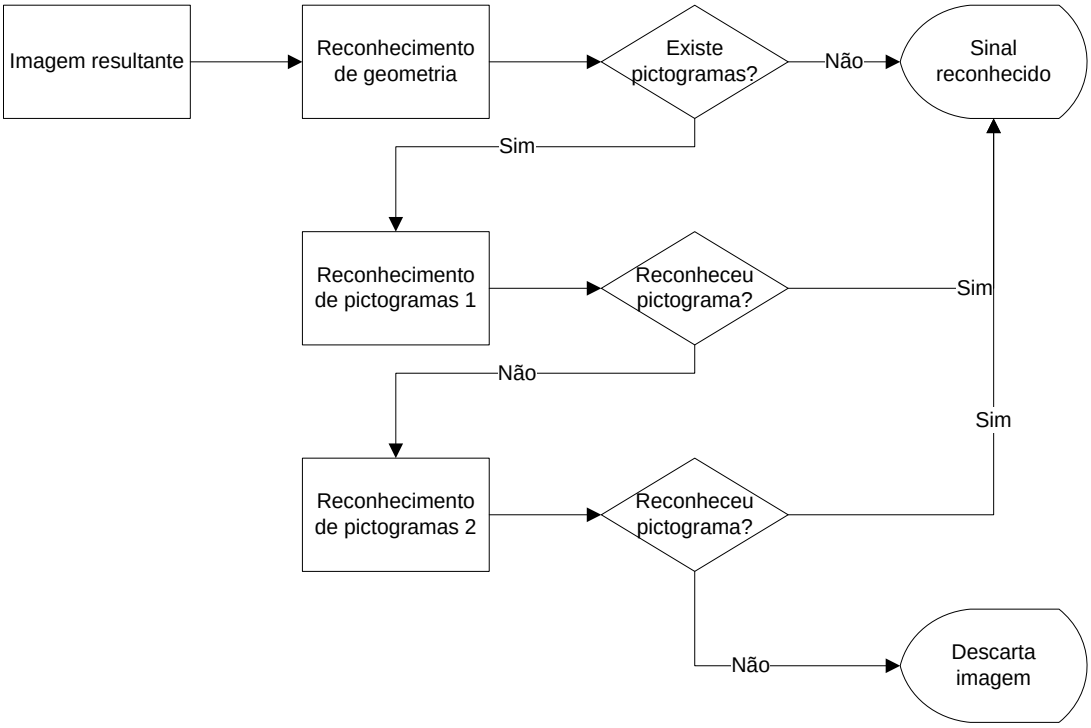


Figura 10 - Fluxograma da classificação

4.5. Comunicação

A comunicação é algo importante aquando do desenvolvimento do dispositivo final. Contudo, não sendo um dos objectivos essenciais deste trabalho, foi utilizada uma comunicação série, face à maior acessibilidade da comunicação com o computador. Os valores comunicados pela câmara são subdivididos conforme exposto na tabela seguinte.

Tabela 3 - Classificação dos sinais de trânsito em classes e valor de retorno

Forma / Cor	△	▽	□	○	STOP
Azul			Informação (6x)	Obrigação (5x)	
Vermelho	Perigo (3x)	Cedência de passagem (200)		Proibição (4x)	STOP (100)

Analisando a tabela anterior, verifica-se que alguns sinais são classificados por um número, enquanto outros por número e caracter. Esta configuração deve-se aos sinais de STOP e

cedência de passagem serem sinais únicos, enquanto que os outros contêm mais informação no seu interior e por isso aparecem com "x", relativo a variação de valores conforme a existência de sinais. A tabela completa de sinais encontra-se no anexo 1 deste documento.

4.6. Nota conclusiva

É importante salientar o facto deste trabalho situar-se na elaboração da fase de classificação, fase esta responsável por detectar, reconhecer e classificar um sinal de trânsito. De realçar também a importância da filtragem da imagem proveniente do sensor CMOS (câmara) para simplificar os processos a jusante.

Capítulo 5 - Código computacional

Introdução

Este capítulo aborda de uma forma pormenorizada todo o estudo realizado. Aborda temas como o recálculo dos parâmetros de HSV, gestão da memória do dispositivo, entre outros. O processo de reconhecimento, sendo um dos processos com maior relevância neste tipo de sistemas, é explicado de uma forma mais detalhada. O capítulo é finalizado com a descrição de como é efectuada a comunicação do sinal previamente obtido.
A forma encontrada para esboçar e dar a entender o trabalho desenvolvido foi a utilização não só de gráficos, mas também de uma estruturação assente em fórmulas.

Para a realização da programação do sistema de detecção e reconhecimento foram utilizados sinais não homologados para treino, e posteriormente, aquando de testes práticos, os valores de cor e forma foram alterados para valores oficiais provenientes do Diário da República (1998) conforme descrito no próximo capítulo.

5.1. Cálculo de HSV

A determinação da cor no plano HSV não foi feita segundo a forma genérica apresentada no sub-capítulo 2.2. Foi realizada uma adaptação das fórmulas (4), (5) e (6) que após a aplicação da fórmula (7) resultou nas fórmulas (10), (9) e (8) respectivamente, isto porque o sistema utiliza 8 bits, valor este que é o provenientes do sensor CMOS. Outro factor a ter em conta reside no acesso ao *buffer* ser feito por colunas e o referido cálculo também ser efectuado em colunas, sequencialmente.

$$novo_{valor} = \frac{valor \times 255}{360} \tag{7}$$

$$V = \max \tag{8}$$

$$S_{\max} = \begin{cases} 0, & \max = 0 \\ 255 \times \dfrac{\max - \min}{valor}, & c.c. \end{cases} \tag{9}$$

$$H_{R,G,B} = \begin{cases} 45 \times \dfrac{G-B}{\max - \min}, & \max = R \\ 45 \times \dfrac{B-R}{\max - \min} + 85, & \max = G \\ 45 \times \dfrac{R-G}{\max - \min} + 171, & \max = B \end{cases} \tag{10}$$

5.2. Gestão de memória

Como anteriormente referido, a reduzida memória do microcontrolador faz com que a programação tenha que ser extremamente optimizada, tal que em cada passagem pela imagem se retire o máximo de informação. Neste caso em particular, os sinais de trânsito são sempre construídos com uma figura geométrica com rebordo azul ou vermelho (excluindo o sinal de via com prioridade e fim de proibição) e a informação dos pictogramas é dada pela cor preta ou pela ausência das 3 cores referidas. Este facto foi considerado devido à limitação de memória volátil.

Foi com base neste pormenor que prosseguiu o desenvolvimento do sistema de detecção e reconhecimento de sinaléticas, realizando a extracção de três imagens binarizadas, isto é, sendo extraídas três imagens que contêm informação de vermelho, azul e preto. Em seguida são apresentados os principais valores de utilização da memória RAM, tendo em consideração que apenas são referidos os valores mais significativos.

$$\begin{aligned} & pilha + bibliotecas + 3 \times imagem_{binarizada} + RAM_{livre} = RAM \\ & 9000 + 9000 + 3 \times (88 \times 144 \times 1) + RAM_{livre} = 64000 \\ & 18000 + 3 \times 12672 + RAM_{livre} = 64000 \\ & 18000 + 38016 + RAM_{livre} = 64000 \\ & RAM_{livre} = 7984 \end{aligned} \tag{11}$$

Desta forma anula-se a possibilidade de detecção de amarelo para incluir o reconhecimento do sinal de via com prioridade. Caso a capacidade de memória e processamento fossem mais elevadas, poderia ser realizado o reconhecimento não só dos sinais de trânsito mais importantes, mas de toda a sinalética, contemplando assim informações sobre as localidades, linhas na estrada, balizas, e outros mais que se julgasse convenientes.

5.3. Processo de reconhecimento

Para o reconhecimento de forma foram analisados alguns documentos científicos tais como Aoyagi e Asakura(1996), Bascón *et al.* (2010), Bueno *et al.* (2005), Carlson *et al.* (2008), Cyganek (2007), Douville (2000), Frias-Martinez *et al.* (2006), Hibi (1996), Hsu e Huang (2000), Krebel *et al.* (1999), Maldonado *et al.* (2007), Miura *et al.* (2000), Paclick *et al.* (2006), Paulo e Correia (2000), Perez e Javidi (2002), Shojania (2003), Song *et al.* (2008), de la Escalera *et al.* (1994, 2003, 2004). A pesquisa do estado da arte forneceu a criatividade para elaborar uma forma própria de reconhecimento, pois as restrições associadas à utilização de recursos muito limitados exigiam a necessidade de se criar um algoritmo feito à medida. Se adicionalmente for considerado que se pretende o desenvolvimento de um sistema de detecção e reconhecimento de sinaléticas em tempo real, então, é necessário um processamento rápido, rápido o suficiente para conseguir processar várias imagens por

segundo, o que faz com que se tenha de encontrar o equilíbrio entre os diversos factores e eliminar todo o excesso.

Neste sentido, as áreas de maior influência para a criação do algoritmo vieram principalmente de estudos sobre o despacho económico (Mariano, 2000), sobre a detecção de rosto desenvolvido por Viola e Jones (2001) e do reconhecimento de sinais de trânsito desenvolvido por Lorsakul e Suthakron (2007) no que diz respeito à utilização do método de Freeman[1].

Assim, foi criada a seguinte sequência de código de modo a processar as imagens:

- ✓ Binarização.
- ✓ Filtragem.
  - Eliminação de ruído.
  - Formas fechadas.
- ✓ Ajuste da imagem
  - Na forma (rotação).
  - No tamanho.
- ✓ Reconhecimento da forma.
- ✓ Reconhecimento de pictogramas.

Em seguida são descritos pormenorizadamente cada um dos parâmetros acima indicados:

BINARIZAÇÃO:
Processo pela qual é criado um conjunto de imagens de 1 bit de cor através da leitura da imagem primária. É de salientar que a mesma (imagem primária) só pode ser lida uma vez do FIFO e que a memória do microcontrolador é insuficiente para o armazenamento dela.

FILTRAGEM:
No ambiente estrada existem diferentes elementos com a mesma cor, isto é, nas imagens resultantes da binarização encontra-se também outra informação (não útil) que terá de ser removida, por isso, é realizada a verificação para comprovar se a forma é fechada. Este processo foi realizado utilizando o método de Freeman (Lorsakul e Suthakorn, 2007), sendo adicionado o factor de incremento e decremento de variáveis x, y e centróide. O processo é finalizado com sucesso se os valores de $\Delta x = 0$ e $\Delta y = 0$, o que significa que o objecto é fechado. Visto ser um pouco complexo textualmente, de seguida é descrito este processo com informação visual.

As seguintes matrizes foram criada para aplicação do método de Freeman, a matriz à esquerda é referente ao valor atribuído enquanto a da direita refere-se às coordenadas onde

[1] Este método tem por base a procura, nas zonas vizinhas, de informação útil. Para tal é construído uma matriz de 3x3 onde o ponto central é o pixel ao qual se está a analisar a vizinhança, o valor é descrito de 1 a 8 consoante a informação da vizinhança.

os valores presentes são $x; y$ e o centro é o ponto de referência, ou seja, o ponto pelo qual se irá proceder à informação da vizinhança.

1	2	3
8		4
7	6	5

-1;1	0;1	1;1
-1;0		1;0
-1;-1	0;-1	1;-1

Figura 11 - Matrizes do método de Freeman modificado

Inicialmente, é calculado o centróide dos elementos juntos para que se possa fazer passar a matriz de Freeman. Isto porque é necessário que se inicie a matriz de Freeman no ponto mais afastado da centróide para evitar erros, um exemplo seria começar por um ponto fixo conforme se demonstra em seguida.

a) b)

Figura 12 - Importância do cálculo do centróide

Nas imagens da figura anterior podem-se analisar o quão importante é o centróide. Na imagem a) fez-se percorrer a matriz de Freeman começando sempre no ponto 0;1 e fazendo um varrimento no sentido horário, ao chegar à extremidade mais afastada ocorre um *looping*, pois a vizinhança é logo encontrada no primeiro ponto. Já com a utilização de um centróide (imagem b)), mesmo que não seja ideal (exactamente no centro da figura) saber-se-á o ponto mais afastado e para que sentido rodar, permitindo com tal percorrer a periferia do sinal.

Após esta operação e caso tenha sido concluída com sucesso, os dados de , , e são armazenados no que se chamam de regiões de interesse (ROI). São distintas nas cores vermelha e azul, contudo o algoritmo está preparado para verificar implicações nas duas ao mesmo tempo, permitindo desta forma analisar sinais de trânsito nomeadamente, estacionamento proibido e paragem (sinais 438 e 439 do anexo 1). Para aumentar a velocidade de processamento e de verificação, foi implementada a verificação de uma possível forma, nomeadamente das formas triangulares, pois verificam a condição:

$$\sum_{x=\min}^{\max} abs(\Delta y) \approx 0 \cup \sum_{x=\max}^{\min} abs(\Delta y) \approx 0 \qquad (12)$$

AJUSTE NA IMAGEM:

Após a fase de filtragem ter-se-ão três imagens binarizadas juntamente com um conjunto de informações de regiões de interesse. Contudo, através da realização de testes preliminares chegou-se à necessidade de proceder à rotação da imagem para prepará-la para a etapa seguinte. Embora se utilize a palavra "rotação" o que realmente sucede é a alteração do passo das variáveis de x e y. Desta forma o que se altera são os pontos de leitura das imagens. À excepção das figuras circulares, todas as outras têm uma região linear, que é sempre encontrada no extremo da figura tendo como partida o centro da imagem (possível após ter sido encontrado o e) e que tem como dimensão 1/3 da largura total. É nesta região que se calculam os valores de passo anteriormente referidos.

Juntamente com este passo é procedido a um ajuste do tamanho de modo a fazer um correcto reconhecimento. O tamanho é ajustado para a dimensão 20x20 pixéis para criar uma imagem amostrada para o passo seguinte (reconhecimento de forma). Este tamanho é ajustado individualmente nas suas coordenadas, conseguindo-se desde modo contornar dois problemas, o de rotação na vertical e o de rotação na horizontal no sinal de trânsito, como mostra na figura seguinte.

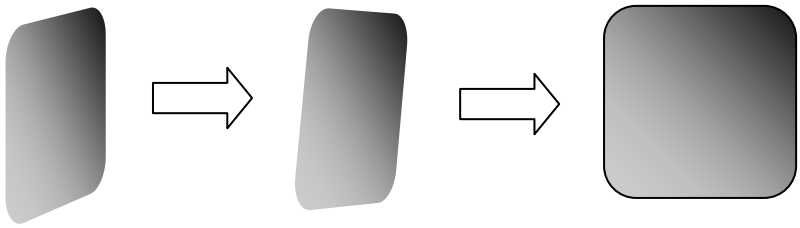


Figura 13 - Exemplo do processo de ajuste da imagem

RECONHECIMENTO DE FORMA:

O reconhecimento de forma foi a fase que mais dúvidas suscitou, uma vez que actualmente já existem bons algoritmos, tal como o utilizado por Paulo e Correia(2000), fazendo uso da utilização de subzonas de interesse. Contudo, os problemas encontrados e referidos pelo investigador foram significativos e após análise de outras referências bibliográficas, verificou-se que também Hsu e Huang (2000) também tinha tido estes mesmos problemas. Assim sendo e no intuito de desenvolver um sistema de detecção e reconhecimento de sinaléticas que se requer mais funcional que complexo, foi realizada uma comparação com uma imagem armazenada, já que a memória ROM do microcontrolador ainda poderia ser utilizada. Foi então aproveitada para armazenar imagens amostra ideais de dimensão 20x20. Desta forma sabe-se ao certo a quantidade de pontos que coincidem, que não coincidem e que não têm correspondência. Posteriormente calculam-se os pesos e quantidades destes aspectos e desta forma calcula-se a probabilidade de correspondência e o facto de enumerar as não correspondências faz com que diminua o reconhecimento de falsos positivos.

RECONHECIMENTO DE PICTOGRAMAS:

O reconhecimento de pictogramas seguiu o mesmo procedimento do reconhecimento de forma, apenas com alteração dos valores dos pesos. Todavia, mesmo assim foram detectadas dificuldades nomeadamente no que diz respeito à aquisição, pois os pictogramas já eram de pouca qualidade e neste aspecto as probabilidades de sucesso foram reduzidas.

5.4. Comunicação

A comunicação não foi um dos factores considerados como prementes no desenvolvimento deste sistema. Foi utilizado o padrão para comunicação série de dados, RS-232, por ser acessível e por haver disponibilidade de material. Devido à frequência e complexidade de um sinal, são considerados dois grupos, um que contém os sinais que normalmente aparecem, isto é, mais vulgares e outro que contém os sinais cuja presença é mais rara. Esta separação permite a diminuição do tempo de processamento.

Um sinal é tido como correctamente classificado quando a sua probabilidade é superior a 90% do valor ideal ou o melhor valor de probabilidade acima de 60% (valores definidos após testes de tentativa e erro). Caso o valor desta probabilidade seja inferior, então parte-se para uma comparação com símbolos que aparecem com menos frequência. Em seguida são apresentadas as duas formas de transmissão destes sinais. Na Equação 13, encontra-se o cálculo do valor do sinal proveniente de sinais frequentemente utilizados, e na Equação 14 encontra-se o método de transmissão do valor do sinal de sinais construídos por forma base e símbolo.

$$sinal_{transmitido} = código_{sinal} + "p" + probabilidade \quad (13)$$

$$sinal_{transmitido} = código_{base} + "s" + código_{símbolo} + "p" + probabilidade \quad (14)$$

Onde:

$sinal_{transmitido}$ – Valor transmitido pela linha.

$código_{sinal}$ – Código do sinal (anexo 1).

$código_{base}$ – Código do sinal da forma (anexo 1).

$código_{símbolo}$ – Código do símbolo (anexo 1).

$"p"$ – Probabilidade.

$"s"$ – Símbolo.

$probabilidade$ – Valor de probabilidade de 0 a 100.

De forma a ser mais fácil a interacção com o utilizador e porque os valores são superiores a 8 bits (tamanho máximo de um pacote de RS-232), todos os valores são transmitidos em forma de caracteres.

5.5. Nota conclusiva

O *software* foi desenvolvido para responder às exigências impostas, contudo também deve ser proporcional à capacidade de processamento e à capacidade de armazenamento do sistema embutido anfitrião. Para tal existe a necessidade de criação de algoritmos simplistas e técnicas de poupança de memória.

Capítulo 6 - Testes experimentais

Introdução

Neste capítulo são apresentados os resultados práticos do trabalho realizado, as áreas de sucesso e áreas a serem melhoradas para o futuro.

Aquando dos testes experimentais foi necessária a pesquisa de uma fonte fiável de sinais de trânsito. Tal foi conseguido através da leitura do Regulamento de Sinalização do Trânsito, aprovado pelo Decreto-Regulamentar n.º 22-A/98, de 1 de Outubro, obtendo-se assim as variáveis de cor, luminância, crominância e forma fazendo com que houvesse uma base sólida de trabalho.

6.1. Valores de cor e sua implementação

Após a análise do referido Decreto-Regulamentar foram obtidas as cores de interesse da seguinte forma:

Tabela 4 - Resumo das cores necessárias segundo o Decreto-Regulamentar

Cores regulamentares	Matiz (H)	Saturação (S)	Luminância (L)
Vermelho	$v < 0,345 \cup v > 0,569$	-	$v \geq 0,03$
Azul	$a > 0,038 \cap a < 0,250$	-	$a \geq 0,01$
Branco	-	-	$b > 0,35$

Onde:
V: valor da cor vermelha.
a: valor da cor azul.
b: valor da cor branca.

A variação dos valores é de 0 a 1. Assim, os filtros a colocar no microcontrolador correspondem a uma proporção directa de 8 bits (255), pelo que os novos valores são os expostos na seguinte tabela.

Tabela 5 - Valores a introduzir no microcontrolador para criação das imagens binarizadas

Cores regulamentares	Matiz (H)	Saturação (S)	Luminância (L)
Vermelho	$v < 88 \cup v > 145$	-	$v \geq 7$
Azul	$a > 9 \cap a < 64$	-	$a \geq 2$
Branco	-	-	$b > 89$

Onde:
V: valor da cor vermelha.
a: valor da cor azul.
b: valor da cor branca.

Após a análise das tabelas, foi detectado um conflito entre a gama de valores da cor azul e da cor vermelha, uma vez que a gama de valores da primeira cor se encontram dentro da gama

de valores da segunda. Contudo, no documento analisado havia valores fixos que foram passados para conjuntos.

Todavia, o facto do espaço de cores ser HSV e no documento referido o valor provir de HSL, para além do sensor CMOS captar as cores de forma diferente da ideal, fez com que fosse necessário realizar uma amostragem para encontrar valores de vermelho, azul e preto.

Os gráficos seguintes mostram a dispersão dos valores das diferentes cores nas componentes H (*Hue* – matiz), S (*Saturation* – saturação) e V (*Value* – valor):

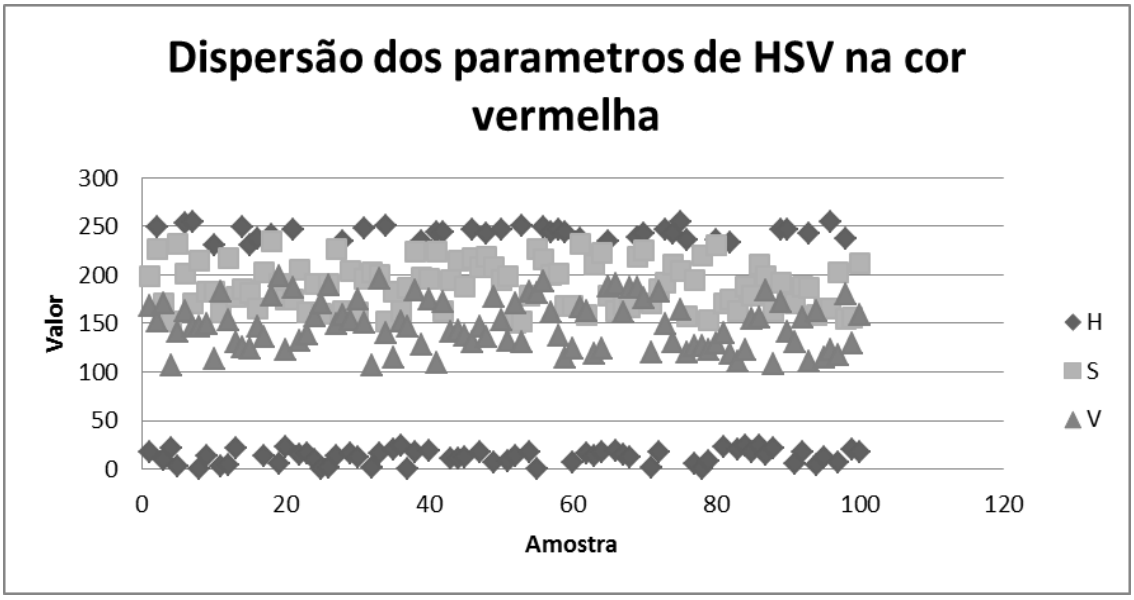


Figura 14 - Gráfico da dispersão dos parâmetros de HSV na cor vermelha

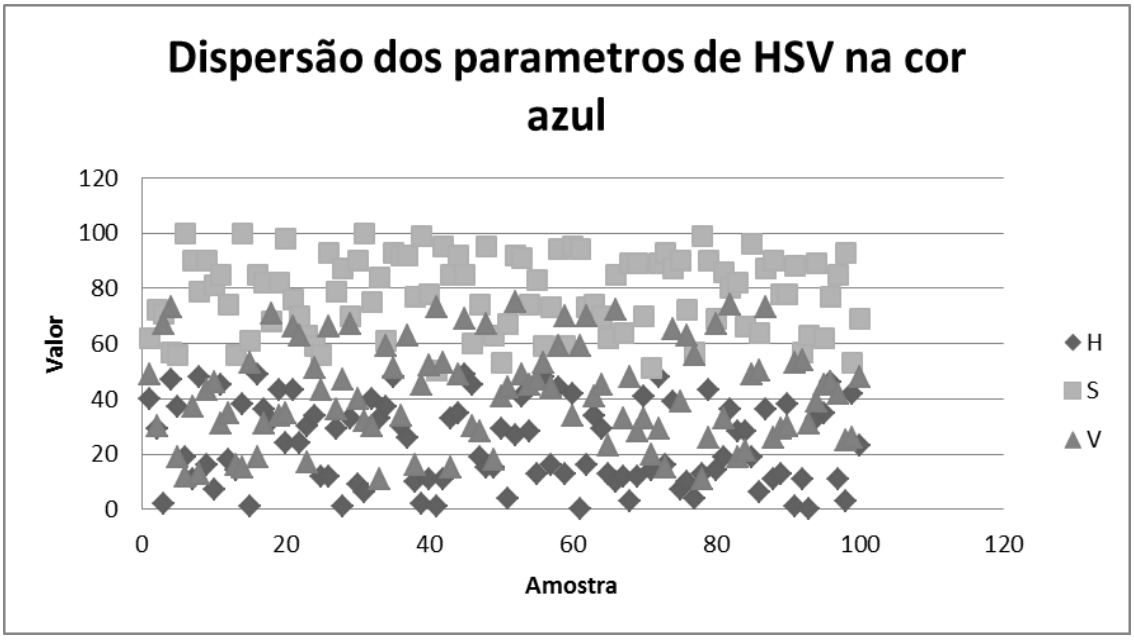


Figura 15 - Gráfico da dispersão dos parâmetros de HSV na cor azul

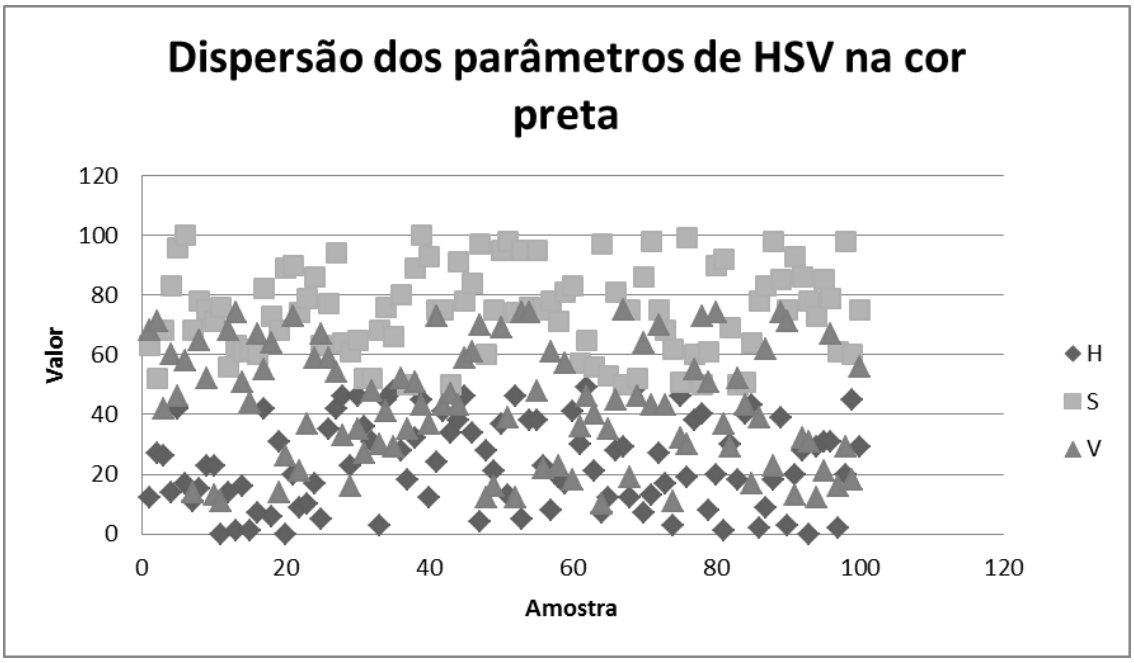


Figura 16 - Gráfico da dispersão dos parâmetros de HSV na cor preta

Os gráficos foram construídos por 100 amostras de sinais de trânsito, visíveis em condições suficientes. A obtenção das imagens foi feita com a CMUcam3 e a obtenção das variáveis foi feito com recurso ao MatLab fazendo uso do código seguinte:

```
1 :    imagem = imread ('imagemX.jpg');
2 :    image(imagem);
```

Figura 17 - Exemplo de código para MatLab

Após a análise dos gráficos e de testes experimentais, obtiveram-se os valores de HSV para as cores vermelha, azul e preta conforme indicado no seguinte quadro resumo:

Tabela 6 - Quadro resumo dos valores de HSV das cores vermelha, azul e preta

Cores regulamentares	Matiz (H)	Saturação (S)	Valor (V)
Vermelho	$v < 30 \cup v > 230$	$v > 150 \cap v < 240$	$v > 70 \cap v < 202$
Azul	$a > 150 \cap a < 220$	$a > 120$	$a < 140$
Preto	-	-	$p < 70$

Onde:
v : valor da cor vermelha.
a : valor da cor azul.
p : valor da cor preta.

Estes valores foram então colocados como filtros para a criação das imagens binarizadas. De realçar que o filtro da cor preta foi construído apenas com a componente de valor de cor,

para diminuir o tempo de categorização e porque era a condição suficiente necessária. De seguida, é apresentado um trecho do código desenvolvido relativo à função de binarização.

```
1 :   void ubi_binarization()  {
2 :     uint8_t h, s, v, y, x, z, i;
3 :     uint16_t read;
4 :     if (depuration == true)                  //mensagem de depuração
5 :       printf ("D inicio ubi_binarization\r");
6 :     if (dub == true)                         //mensagem de depuração
    extra
7 :       printf ("E row a ser realizado\r");
8 :     uint8_t *row = cc3_malloc_rows (1);      //apontador para conversão
    da imagem
9 :     if (dub == true)                         //mensagem de depuração
    extra
10 :      printf ("E row realizado\r");

11 :    pixeis[0] = 0;                        //inicialização do nº pixéis de
    cada cor
12 :    pixeis[1] = 0;
13 :    for (y = 0; y < altura; y++) {         //correr valores de y
14 :      cc3_pixbuf_read_rows (row, 1);       //carregar coluna para
    converter em HSV
15 :      cc3_rgb2hsv_row(row, largura);       //converter coluna para HSV
16 :      for (x = 0; x < largura; x++) {      //correr valores de x
17 :        for (z = 0; z < 3; z++) {          //para separar H, S e V
18 :          read = 3 * x + z;
19 :          if (z==0)
20 :            h = row[read];
21 :          if (z==1)
22 :            s = row[read];
23 :          if (z==2)
24 :            v = row[read];
25 :        }                     //obtenção de HSV, segue-se a
    classificação
26 :        if ((h > 230 || h < 30) && s > 150 && s < 240 && v > 70 && v <
    202){  //vermelho
27 :          imagem[x][y][1] = true;
28 :          pixeis[0] = pixeis[0] + 1;
29 :        }
30 :       else
31 :         imagem[x][y][1] = false;
32 :        if (h > 150 && h < 220 && s > 120 && v < 140){
    //azul
33 :          imagem[x][y][2] = true;
34 :          pixeis[1] = pixeis[1] + 1;
35 :        }
36 :        else
37 :          imagem[x][y][2] = false;
38 :        if (v < 70
    //preto
39 :          imagem[x][y][3] = true;
40 :       else
41 :         imagem[x][y][3] = false;
42 :      }
43 :    }
44 :    free (row);
45 :    if (depuration == true)        //mensagem de depuração
46 :      printf ("D fim ubi_binarization\r");
47 :  }
```

Figura 18 - Código desenvolvido para realizar a binarização da imagem

No código da figura anterior existem duas zonas de maior importância, a primeira relativa à conversão da imagem RGB para HSV. Esta é realizada pelo código presente na linha 15,

através de um algoritmo fornecido com a câmara (*cc3_rgb2hsv_row(…)* auxiliado pela função *cc3_pixbuf_read(…)*). Posteriormente, da linha 17 à 25, são separadas as três componentes (H, S e V) para se dar inicio à selecção que ocorre entre nas linhas 26-42. Neste mesmo código, nas linhas 11-12, 28 e 34, recorre-se a uma variável para enumerar a quantidade de pixéis (*pixéis[0:1]*) de cada cor encontrados. Desta forma é possível inserir um valor mínimo (*threshold[0:1]*) para iniciar as etapas seguintes. O processo de binarização é feito em cascata, permitindo desta forma aumentar a velocidade de classificação da cor, isto porque, caso a cor seja classificada como vermelha, não poderá ser outra das cores e assim sucessivamente. Finalmente existe um conjunto de mensagens que foram utilizadas para a depuração de cada função, separadas em duas. A primeira serve para saber quando o algoritmo foi iniciado e finalizado (retornando o valor de fim se possível, neste caso, este valor não existe pois a função não retorna valores). Esta mensagem inicia-se com a letra D e tem como variável global *depuration*. O outro tipo de mensagens são as "extra" de cada algoritmo. Assim, a variável de controlo é específica para cada algoritmo. Neste caso é a *dub* (depuração do ubi_binarization()), sendo a mensagem apresentada no computador iniciada pela letra E.

6.2. Teste laboratorial

No dispositivo foi implementado um modo de depuração que torna possível exportar as imagens binarizadas. Foi então criado um programa em MATLAB para que pudesse receber essas imagens.

A utilização do MATLAB surgiu por ser um programa familiar e por ter as características necessárias para o desenvolvimento, nomeadamente no recurso a um ambiente gráfico (*Graphical User Interface* - GUI) e acesso ao porto de comunicações. Em seguida é apresentado o resultado.

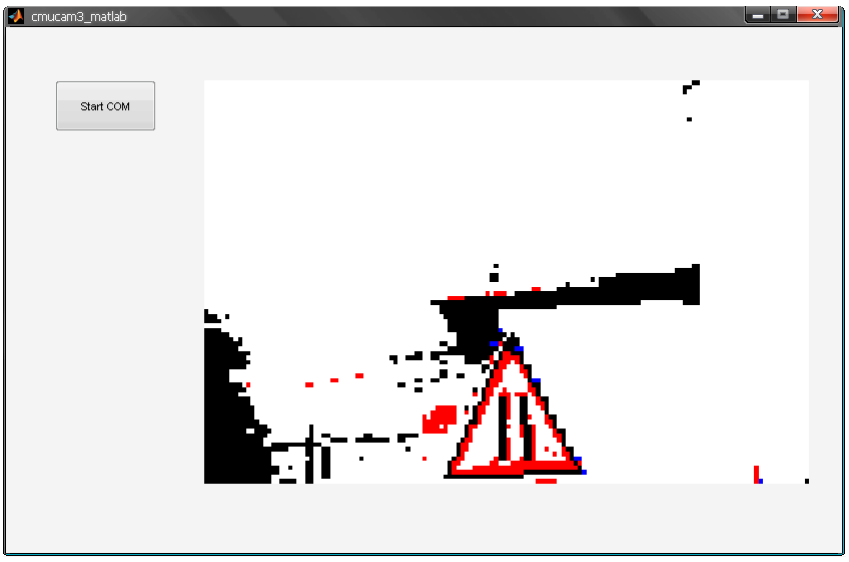


Figura 19 - Interface gráfica do MatLab.

Embora apenas seja visível a imagem multi-binarizada, o programa recebe e guarda as imagens provenientes da câmara, nomeadamente a imagem da forma, pictograma preto e pictograma branco, conforme serão mostrados ao longo deste capítulo. O código para a interface é demonstrado em seguida:

```
1 :    function start_com_Callback(hObject, eventdata, handles)
2 :    imagem = imread ('base.bmp');   //imagens base, para serem modificadas
3 :    imagem_forma = imread ('20x20_preto.bmp');
4 :    imagem_preto = imagem_forma;
5 :    imagem_branco = imagem_forma;
6 :    s = serial('COM1');   //obrigatoriamente tem de estar na COM1, se for
    para exportação deste código fazer uma mensagem pop-up
7 :    s.InputBufferSize = 50000;  //definir um valor muito grande para nunca
    esgotar
8 :    s.BaudRate = 115200;  //valor sempre fixo (baudrate = 115200 = cmucam3)
9 :    s.Terminator = 'CR';  //termina a string com um enter
10 :   fopen(s);  //inicia a ligação
11 :   A=fscanf(s);  //para saltar o enter da string que a camara manda com o
    nome do autor e universidade
12 :   i=true;
13 :   a=true;
14 :   b=true;
15 :   while (i)
16 :     line=fgetl(s)   //adquire os dados em strings (até encontrar o enter)
    --> este bocado não pode ser obtido de outra forma, é preciso a captura
    mesmo por strings completas senão falha dados visto que é comunicação
    assíncrona.
17 :     if (strcmp(line(1),'I')==true) //se a string começar com I é imagem
    multi-binarizada (grande)
18 :       a=false;
19 :       for z=1:3 //coordenada z (profundidade)
20 :         for y=1:88 //coordenada y (vertical)
21 :           for x=1:144  //coordenada x (horizontal)
22 :             if(line(x+144*(y-1)+144*88*(z-1)+1)=='1')
23 :               if (z==1)  //na cmucam 1 = vermelho
24 :                 imagem(y,x,1)=255;  // no RGB, vermelho é profundidade 1
25 :                 imagem(y,x,2:3)=0;
26 :               end
27 :               if (z==2)  //na cmucam 2 = azul
28 :                 imagem(y,x,1:3)=0;
29 :                 imagem(y,x,3)=255;  //no RGB, azul é profundidade 3
30 :               end
31 :               if (z==3)  //na cmucam 3 = preto
32 :                 imagem(y,x,:)=0;  //no RGB, preto é 0 em todos
33 :               end
34 :             end
35 :           end
36 :         end
37 :       end
38 :     end
39 :     if (strcmp(line(1),'J')==true) //se a string começar por J é imagem
    multi-redimensionada (3 pequenas)
40 :       b=false;
41 :       for z=1:3
42 :         for y=1:20
43 :           for x=1:20
```

```
44 :             if(line(x+20*(y-1)+20*20*(z-1)+1)=='1')
45 :               if (z==1)
46 :                 imagem_forma(y,x) = true;
47 :               end
48 :               if (z==2)
49 :                 imagem_preto (y,x) = true;
50 :               end
51 :               if (z==3)
52 :                 imagem_branco (y,x) = true;
53 :               end
54 :             end
55 :           end
56 :         end
57 :       end
58 :     end
59 :     if (a == false && b == false)
60 :       i = false;
61 :     end
62 :   end
63 :   axes(handles.axes1);
64 :   image(imagem);
65 :   datacursormode on;
66 :   fclose(s)  //fecha ligação aqui
67 :   delete(s)
68 :   clear s
69 :   imwrite(imagem,'imagem.bmp','bmp'); //guardar imagens todas
70 :   imwrite(imagem_forma,'resize_forma.bmp','bmp');
71 :   imwrite(imagem_preto,resize_preto.bmp','bmp');
72 :   imwrite(imagem_branco,resize_branco.bmp','bmp');
```

Figura 20 - Código da interface de MatLab.

Na figura anterior encontra-se o código responsável por adquirir as imagens a partir da câmara. De salientar que a câmara está a funcionar em ciclo infinito, isto é, está sempre a enviar novas imagens fazendo com que o MatLab tenha de se "sincronizar" com esta (o que acontece nas linhas 17 e 39). Esta condição sucede quando a câmara envia o carácter I, ou seja, inicio da imagem multi-binarizada, ou J, inicio das imagens multi-reamostradas. Após receber as imagens dá por finalizado o processo, exibindo a imagem multi-binarizada e guardando todas as imagens. Nas linhas 2-5 são iniciadas as variáveis com imagens pré-definidas, enquanto que nas linhas 6-9 é definido a porta de comunicações e respectivo *buffer* de entrada. As imagens são construídas entre as linhas 15-62 e guardadas nas linhas 69-72. De realçar que em situação normal este programa não irá funcionar com a câmara, uma vez que esta não irá emitir as imagens mas apenas os códigos dos sinais de trânsito classificados conforme demonstrado ao longo deste capítulo.

Os resultados dos testes laboratoriais são apresentados nas imagens seguintes.

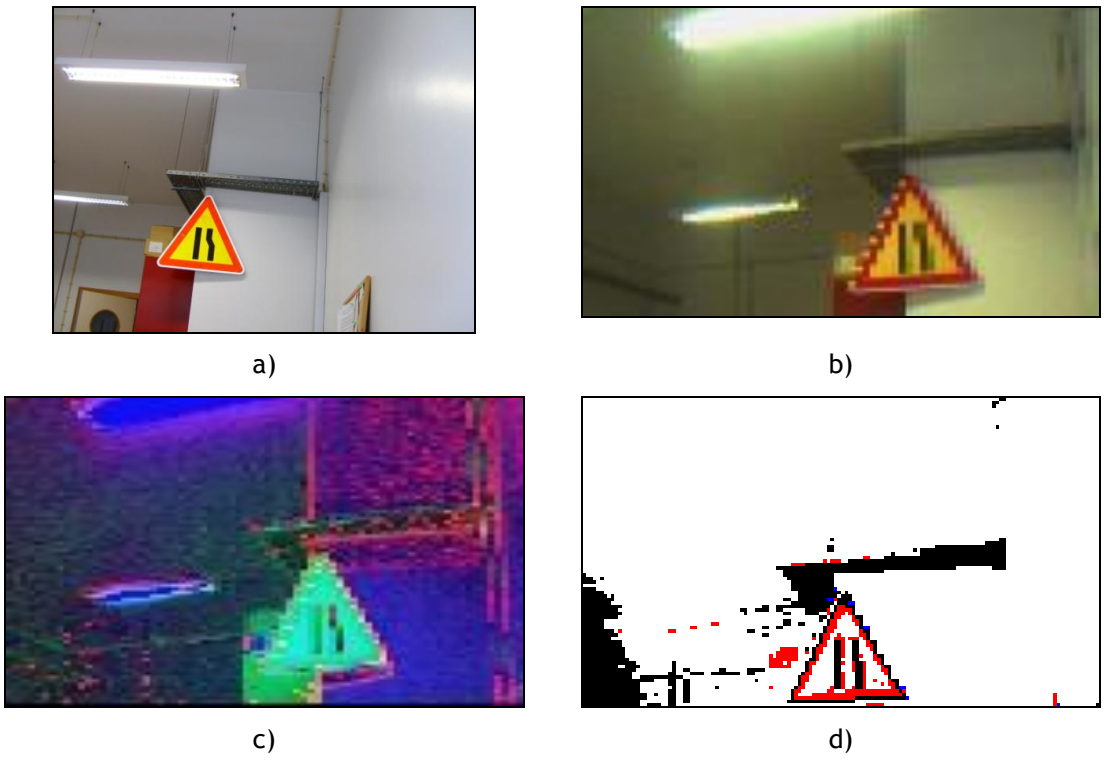


Figura 21 - Comparação entre imagens reais e imagem binarizada.

Na imagem a) da figura anterior é apresentada o contexto espacial onde foi realizado o teste. A imagem é proveniente de uma máquina fotográfica com uma resolução de 5 MP (megapixéis). A imagem b) e c) da mesma figura são respectivamente os formatos RGB e HSV provenientes do programa CMUcam2GUI.jar (programa facultado pela Carnegie Mellon University com a câmara CMUcam2 e compatível com a CMUcam3). A imagem d) é uma imagem multi-binarizada importada para o programa desenvolvido em MATLAB (anteriormente demonstrado) de forma a facultar informações de depuração ao programador. Nesta mesma imagem podem-se analisar as 3 imagens (vermelho, azul e preto) que são formadas pela binarização. Adicionalmente, também é demonstrada a análise do algoritmo de obtenção de forma (linha preta que contorna o sinal triangular).

Um excerto do algoritmo responsável pela criação do limite da forma é apresentado em:

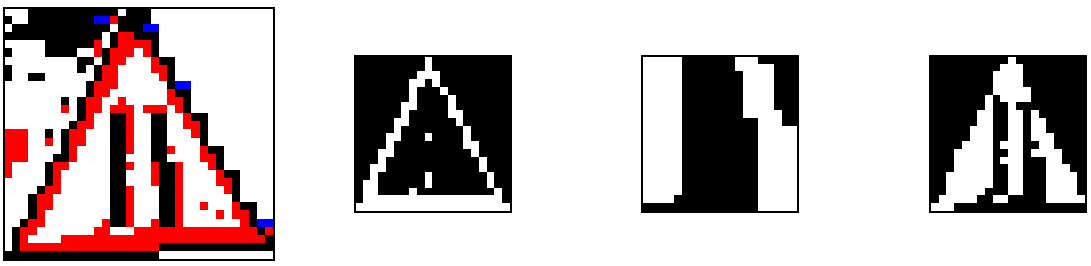
```
1 :    for (fi = 0; fi < 8; fi++){
2 :      …
3 :      if (dugfi==true)
4 :        printf("E %dªanalise (%d) - Analisando ponto %d %d  (%d %d)\r", fi,
    fp, (x + UBIFREEMAN[fp-1][0]), (y + UBIFREEMAN[fp-1][1]), UBIFREEMAN[fp-
    1][0], UBIFREEMAN[fp-1][1]);
5 :      …
6 :      if(imagem[(x  +  UBIFREEMAN  [fp  -1][0])][(y  +  UBIFREEMAN  [fp-
    1][1])][seleccao+1] == true && imagem[(x + UBIFREEMAN [fp -1][0])][(y +
    UBIFREEMAN [hp-1][1])][3] == false){
7 :        x = x + UBIFREEMAN [fp -1][0];
8 :        y = y + UBIFREEMAN [fp -1][1];
9 :        fi = 10;
10 :       imagem[x][y][3] = true;
11 :       if (dugfi == true)
12 :         printf("E ENCONTRADO em %d %d\r", x, y);
13 :     }
14 :   }
```

Figura 22 - Excerto de código do método de Freeman

Após ter sido previamente calculado o melhor valor de início, demonstrado anteriormente, o algoritmo percorre a matriz UBIFREEMAN, matriz esta que contém os valores relativos ao método de Freeman. De modo a não repetir o percurso já realizado, este é gravado na imagem preta (visto não ser possível haver cores que sejam simultaneamente vermelho/azul e preto) e deste modo é possível a visualização do contorno na recepção da imagem multi-binarizada como poderá ser visto ao longo deste capítulo.

É de salientar que todas as imagens colocadas nesta secção têm uma linha de contorno a preto de modo a delimitar as imagens que são proporcionais entre si, com o intuito de poder dar ao leitor a percepção correcta.

Posteriormente a este passo, são obtidas as três imagens de amostragem. Uma vez mais com recurso a MATLAB foi possível a importação das mesmas.



a) b) c) d)

Figura 23 - Imagens amostradas.

Na figura 23 são apresentadas as três imagens amostradas (b), c) e d)). Estas são criadas a partir da imagem a) pelo algoritmo “puxa orelhas”. O algoritmo “puxa orelhas”, através da delimitação da imagem a), adquire as características necessárias para criar a imagem b) pela

componente da forma (neste caso vermelho) que esteja delimitado pela linha proveniente do método de Freeman, a imagem c) pela componente preta e a imagem d) pelas restantes componentes (neste caso não preto e não vermelho, ou seja, branco e azul) não fazendo distinção se o sinal tem um fundo branco ou amarelo tal como demonstrado neste exemplo (o fundo é amarelo, contudo o algoritmo processa-o da mesma forma como se fosse branco). As imagens b), c) e d) são todas de dimensões 20x20 pixéis.

Após a criação de todas as imagens amostradas procede-se a uma comparação destas, inicialmente na forma, tal como é descrito em seguida.

a) b)

Figura 24 - Comparação entre formas

Tal como descrito anteriormente, cada forma é guardada na memória ROM e tem 2 bits, indicando em que cor se aplica (se se aplica em vermelho, se em azul, se em ambas). Neste caso, as formas comparadas são as de STOP, triângulo (sendo que triângulo e triângulo invertido são iguais excepto a sua forma de leitura) e círculo. Da figura 19, a imagem a) é guardada em memória, e a imagem b) é obtida pelo processamento. Os resultados foram os seguintes:

Tabela 7 - Tabela exemplo para encontrar a melhor forma

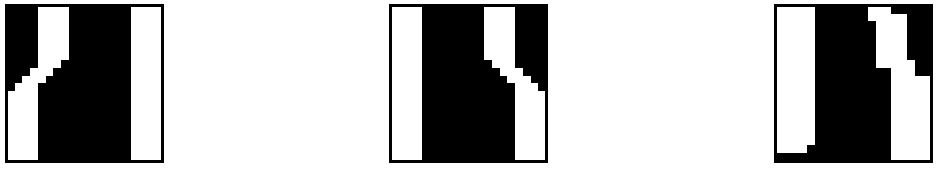
Forma	Vermelho	Azul	Bit coincidente (i)	Bit não coincidente (ii)	Bits (iii = i-ii)	Bits totais (iv)	Probabilidade (iii / iv)
STOP	Sim	Não	56	31	25	273	9%
Triângulo	Sim	Não	83	4	79	140	56%
Triângulo Invertido	Sim	Não	17	70	-53	140	(-)38%*
Circulo	Sim	Sim	27	60	-33	144	(-)23%*
Quadrado	Não	Sim	-	-	-	-	-

*Valores não calculados. A partir do momento em que os Bits são negativos não é calculada a Probabilidade.

Na tabela anterior pode-se observar não só o intervalo da diferença de valores que existe entre a forma certa e as outras formas possíveis, mas também a aplicação dos bits que activam o processamento da forma, neste caso a forma quadrado não foi processada pois é uma forma apenas azul.

De seguida é processado o pictograma que forma suporta a forma (p. ex., o triângulo apenas suporta informação de pictograma preto, o STOP e o triângulo invertido não têm qualquer pictograma, etc...). Neste método foi adicionado uma característica ao processamento do pictograma, fica em memória o maior valor desde que superior a 60%, sendo este substituído por outro de maior probabilidade que é dado como certo caso seja superior a 90%. Os valores de 60% e 90% são provenientes de tentativa erro. Este método foi assim implementado para que fosse reduzido o tempo de processamento, pois os pictogramas também têm bits que indicam o código de saída para cada forma (se possível). Os códigos dos pictogramas encontram-se nos anexos 1 e 2, e foram utilizados uma vez mais para diminuir o tempo de processamento e com o intuito de determinar se a imagem tem uma correspondente depois de proceder a uma inversão na horizontal (as imagens que permitem são colocadas nos primeiros registos).

A imagem seguinte é um exemplo disso.



a) b) c)

Figura 25 - Exemplo de comparação entre pictogramas

Na figura anterior, a imagem b) é proveniente da inversão na horizontal da imagem a) (imagem guardada em memória não volátil) e a imagem c) é a imagem amostra do pictograma (informação da cor preta) adquirido depois de realizado o processamento da imagem. Neste caso em particular, e fazendo uma análise aos pictogramas vizinhos, os valores foram os seguintes:

Tabela 8 - Exemplo do processamento dos pictogramas.

Pictograma	Bit coincidente (i)	Bit não coincidente (ii)	Bits (iii = i-ii)	Bits totais (iv)	Probabilidade (iii / iv)	Triângulo Vermelho (3xx)	Círculo Vermelho (4xx)	Círculo Azul (5xx)	Quadrado Azul (6xx)	Pictograma
	102	83	19	156	12%*	311	-	-	-	-
	145	40	105	156	67%	312	-	-	-	-
	5	180	-175	34	(-)514%*	315	-	-	-	-

*-O valor não é calculado. A partir do momento que o ponto Bits é inferior a 60% não é procedido ao restante cálculo. O valor de Bits mínimo é guardado juntamente com a imagem do pictograma em memória não volátil.

Desta forma, sabendo que foi reconhecida a forma de um triângulo, basta efectuar a leitura na coluna do triângulo vermelho. Existindo uma correspondência, o valor de saída é o código 312 correspondente ao pictograma do sinal de trânsito de Perigo relativo a "passagem estreita: indicação de um estreitamento da via". Analisando a fórmula (13), o valor de saída, como comprovado, foi o seguinte:

$$sinal_{recebido} = 312 + "p" + 67 < cr >$$

Onde:

$sinal_{recebido}$ = sinal recebido pelo computador.

$"p"$ = caracter p.

<cr> = *carrier return* (enter).

Caso o passo seja terminado com sucesso, a imagem é eliminada em todas as camadas e procede-se novamente a uma pesquisa na cor vermelha. Após insucesso da cor vermelha procede-se à cor azul. Se esta não tiver qualquer informação útil é descarregada outra imagem do *buffer* de imagem.

Embora pouco contemplado ao longo deste documento mas de fundamental importância é o ficheiro que contém as informações sobre as formas e sinais (fes.h) que foi criado com recurso ao MatLab pelo código que se segue:

```
1 :    fid = fopen('sinal.h', 'w');
2 :    fid2 = fopen('prob.h', 'w');
3 :    fwrite(fid,'const bool PIC1[102][20][20] = {');
4 :    fwrite(fid,13);
5 :    fwrite(fid2,'const uint8_t PIC1PROB[102][3] = {');
6 :    fwrite(fid2,13);
7 :    for k=0:101
8 :      fwrite(fid,'{');
9 :      str = sprintf('%03.0f.gif',k);
10 :     A = imread (str);
11 :     B = imread('stop.bmp');
12 :     xmin=255;
13 :     xmax=0;
14 :     ymin=255;
15 :     ymax=0;
16 :     [altura,largura,profundidade] = size(A);
17 :     for i=1:altura
18 :       for j=1:largura
19 :         if (A(i,j)~=0)
20 :           A(i,j)=255;
21 :         else
22 :           if (xmin>j)
23 :             xmin=j;
24 :           end
25 :           if (xmax<j)
26 :             xmax=j;
27 :           end
28 :           if (ymin>i)
29 :             ymin=i;
30 :           end
31 :           if (ymax<i)
32 :             ymax=i;
33 :           end
34 :           A(i,j)=0;
35 :         end
36 :       end
37 :     end
38 :     C = imresize(A(ymin:ymax,xmin:xmax), [20 20]);
39 :     pixel=0;
40 :     for i=1:20
41 :       fwrite(fid,'{');
42 :       for j=1:20
43 :         if (C(i,j)~=0)
44 :           B(i,j)=false;
45 :           if (j<20)
46 :             fwrite(fid,'false,');
47 :           else
48 :             fwrite(fid,'false');
49 :           end
50 :         else
51 :           B(i,j)=true;
52 :           pixel=pixel+1;
53 :           if (j<20)
```

```
54 :             fwrite(fid,'true,');
55 :           else
56 :             fwrite(fid,'true');
57 :           end
58 :         end
59 :       end
60 :       if (i<20)
61 :         fwrite(fid,'},');
62 :         fwrite(fid,13);
63 :       else
64 :         fwrite(fid,'}');
65 :       end
66 :     end
67 :     d(k+1,1)=pixel;
68 :     str2 = sprintf('{%d, %d, %d},',pixel, round(0.6*pixel),
    round(0.9*pixel));
69 :     fwrite(fid2,str2);
70 :     fwrite(fid2,13);
71 :     fwrite(fid,'},');
72 :     fwrite(fid,13);
73 :     str2 = sprintf('1%03.0f',k);
74 :     imwrite(B,str2,'gif');
75 :   end
76 :   fwrite(fid,13);
77 :   fwrite(fid,'};');
78 :   fclose(fid);
79 :   fwrite(fid2,'};');
80 :   fclose(fid2);
81 :   xlswrite('sinais&corr', d);
```

Figura 26 - Código de criação das características de pictogramas

No código apresentado na figura anterior é demonstrado como são construídas as matrizes que contêm os pictogramas, neste caso, os pretos. Este programa tem a particularidade de criar um ficheiro e a partir das imagens criar automaticamente a matriz de dados, o que seria muito difícil de efectuar manualmente. Como resultado deste código, são criadas duas matrizes, uma contendo a informação sobre o pictograma, outra contendo os valores das probabilidades que são utilizados para guardar o valor ou dar como certo, processo este explicado anteriormente. Nas linhas 1-6, o código cria os dois ficheiros que irão conter as informações. Posteriormente inicia a criação das matrizes (de modo a que possa ser lido pela CMUcam3). O passo seguinte consiste na leitura de cada imagem e retirar a componente preta, isto é, realizar o algoritmo "puxa-orelhas" (linha 17-38). Para este caso é simples visto as imagens estarem em condições perfeitas, ou seja, basta encontrar o intervalo onde se encontram pixéis pretos e realizar um reacondicionamento espacial da imagem para uma imagem de 20x20 pixéis seguido da sua binarização e guardar no ficheiro, o que acontece entre as linhas 40 e 66. No passo seguinte são calculados os 60% e 90% dos pixéis, valores anteriormente explicados, e guardados no respectivo ficheiro. Também existe o armazenamento dos valores totais de pixéis num ficheiro Excel por motivos de testes extraordinários.

Para este caso em particular, o tempo necessário para processar a imagem é descrito no gráfico seguinte:

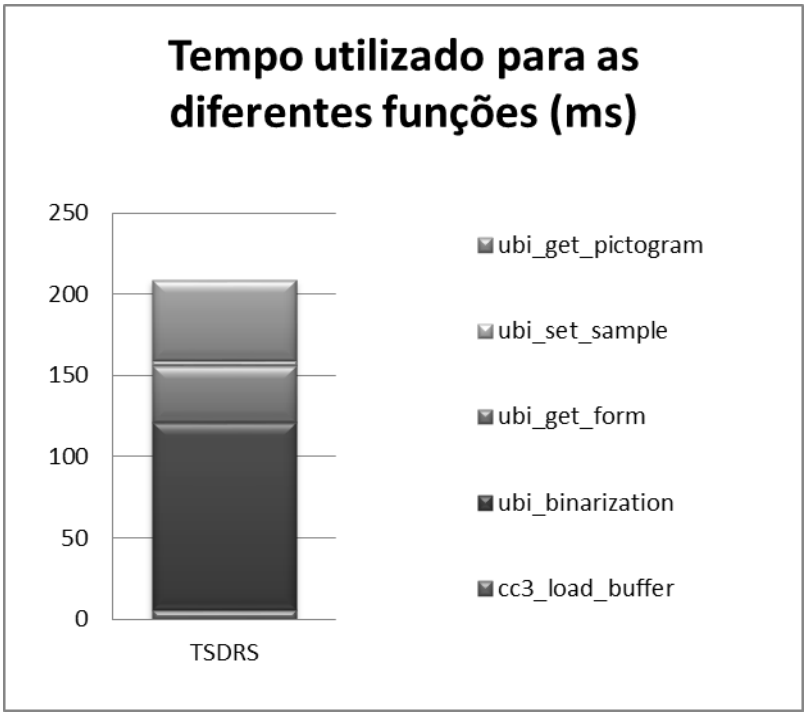


Figura 27 - Exemplo do tempo dispendido para o reconhecimento de um sinal de trânsito

Este gráfico é proveniente dos tempos contabilizados pelo microcontrolador para realizar as diferentes funções sendo elas:

- cc3_load_buffer – Função responsável por preparar o *buffer* para descarregar uma imagem. Despendeu sempre o valor constante de 6 ms.
- ubi_binarization – Responsável pela construção das imagens binarizadas. Internamente tem a capacidade de adquirir os valores de RGB do *buffer*, converter para HSV e realizar a classificação. Por este facto, é a função que ocupou mais tempo de processamento num total de 115 ms. Este valor é praticamente constante.
- ubi_get_form – É a função que verifica se os pixéis têm características de forma fechada e a que forma pertence. O processo para este exemplo demorou 35 ms, pelo que deve ser multiplicado por cada região analisada.
- ubi_set_sample – Função com a responsabilidade de aplicar o algoritmo "puxa orelhas" e construção das diferentes sub-imagens como exemplificadas anteriormente. Demorou 3 ms, valor este que dependerá de quantas sub-imagens serão necessárias criar.
- ubi_get_pictogram/2 – Funções com o objectivo de comparar as imagens amostra com os diferentes registos. Esta função terá tempos muito dispares uma vez que poderá encontrar nos primeiros registos o pictograma, como poderá apenas encontrá-la no final. O registo é composto por 103 registos para pictogramas mais comuns e 30 para os restantes o que faz um total de 133 pictogramas. Neste exemplo, o pictograma foi

encontrado em 50 ms, até porque não houve a necessidade de pesquisar nos pictogramas auxiliares, o que demoraria mais 13 ms (valor proveniente de testes).

- Finalmente a função ubi_set_erase tem como objectivo eliminar a imagem reconhecida, o que fez em 1 ms.

Para este exemplo, o sinal de trânsito foi detectado, reconhecido e classificado em 210 ms, o que faz com que o sistema tenha a capacidade de processar a 4,8 FPS (*frames per second* – Imagens por segundo). Este valor é apenas meramente exemplificativo, podendo o sistema ter valores de tempo mais pequenos caso não existam pixéis vermelhos ou azuis, ou demorar mais tempo caso haja vários sinais de trânsito e/ou pictogramas em registos superiores.

6.3. Resumo dos testes

Os testes laboratoriais provaram ser adequados, contudo, nestes pôde-se analisar algumas falhas, nomeadamente, no que toca à falta de informação no sinal, ou ruído no pictograma de cor branca (que neste caso é a ausência das 3 cores). Em seguida é mostrado uma série de imagens onde se pode analisar estas falhas.

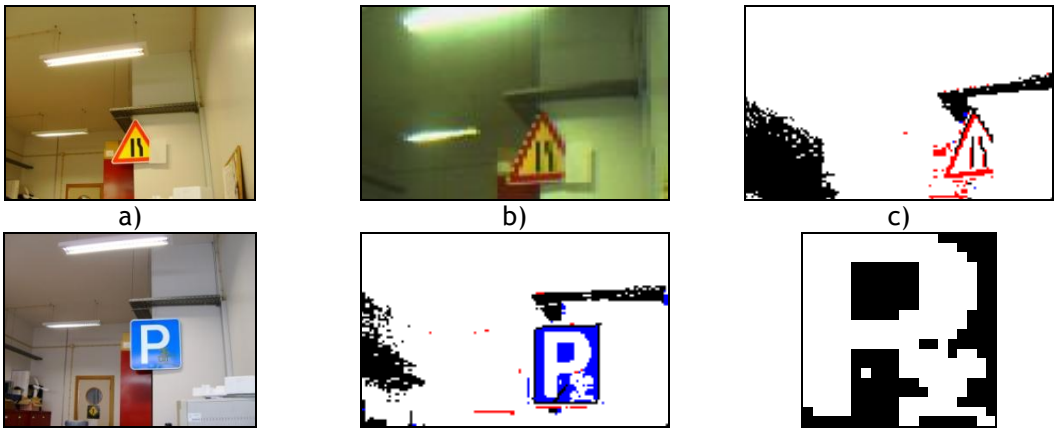


Figura 28 - Exemplo de casos de insucesso.

No caso da figura anterior temos dois conjuntos de imagens compostos pelo conjunto a), b) e c) e o outro conjunto é o d), e) e f). No primeiro conjunto é simulado a falha parcial do sinal. Este é descartado pois o algoritmo não tem capacidade de determinar a forma da figura, pois não a consegue contornar. Analisando a imagem c) nota-se um ligeiro traço a contornar a figura, porém este termina na falha do sinal fazendo com que não tenha sucesso a aquisição da forma. No segundo conjunto é simulado um outro problema frequente relacionado com a introdução de ruído, o que na realidade poderá ser proveniente da degradação do sinal ou mesmo de actos de vandalismo. Contudo, neste exemplo é demonstrado que o pictograma não é devidamente amostrado pois o ruído aumenta a área de interesse. Este facto é mais relevante na cor branca, pois esta é analisada pela ausência das 3 cores (vermelho, azul e preto). Esta situação pode fazer com que o sinal seja erroneamente caracterizado embora

consiga adquirir a forma como se pode notar pela linha preta que percorre a forma na imagem e) da figura anterior.

O ambiente simulado foi pensado de modo a poder abranger diferentes problemas, com intuito de melhorar as características de reconhecimento do algoritmo "puxa orelhas". Posteriormente partiu-se para a estrada onde foram registadas observações para quando da sua implementação final, sejam tidas em conta as seguintes necessidades:

- ✓ Ter em atenção a luz do sol de elevada luminosidade, nomeadamente com incidência directa no sensor de imagem.
- ✓ Pode e deve ser colocado num local ao alcance das escovas de limpeza do pára-brisas, de modo a que o vidro esteja sempre o mais limpo possível.
- ✓ O sensor de imagem não consegue adquirir imagens em ambiente com luminosidade reduzida.
- ✓ O material envolvente não deve ser reflector de modo a que não provoque luz de fundo que posteriormente será reflectiva no vidro.

Infelizmente aquando dos testes reais foi comprovado que o sensor de imagem não tinha capacidades para uma utilização nocturna. Foram indagadas formas para contornar o problema, contudo não é possível dirigir as luzes para cima (pelo perigo de encandeamento dos outros condutores) e não é possível ter um foco de infravermelhos, pois a binarização necessita das cores para realizar as imagens binarizadas.

Após todas estas observações, a implementação final do sistema é dada por:

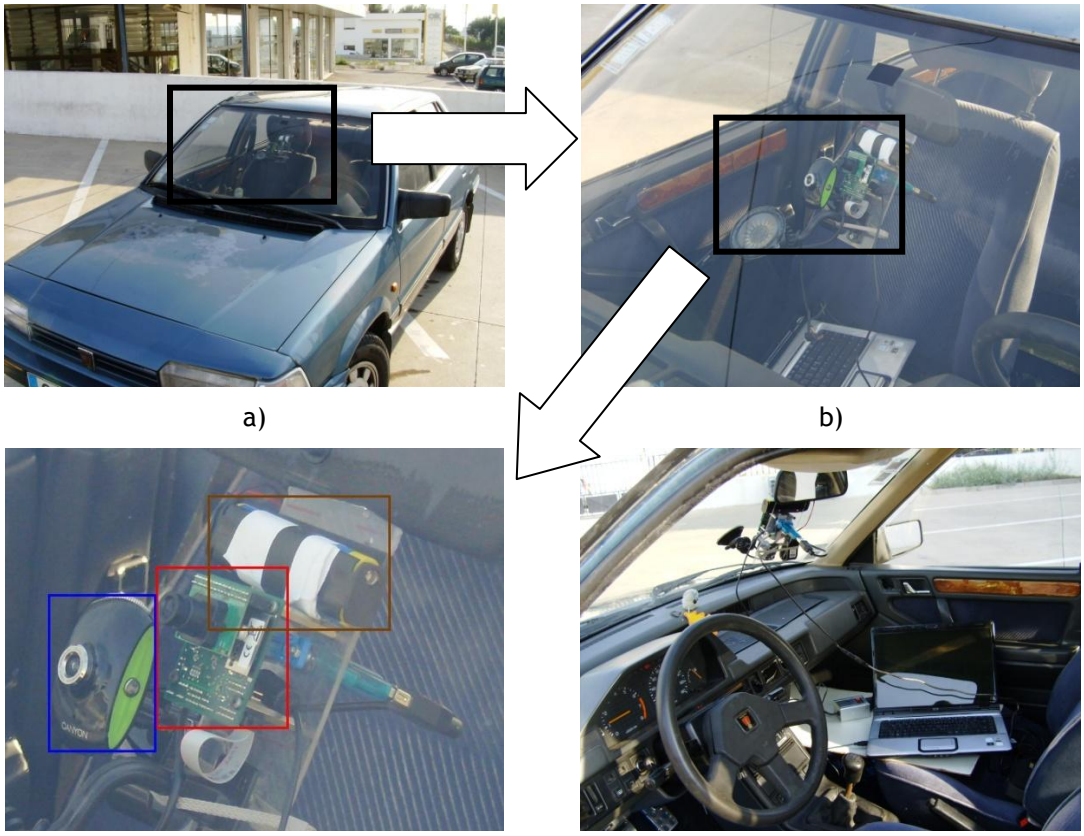


Figura 29 - Imagens da montagem de simulação

Na figura anterior é demonstrado o aspecto da colocação do material para a simulação dos testes reais. Na imagem a) é mostrado o sistema numa vista geral pormenorizando-se até à figura c) onde é mostrado, da esquerda para a direita, a webcam (azul) de forma a adquirir a imagem que a CMUcam3 visualiza (aproximadamente). Ao centro encontra-se a CMUcam3 (vermelho), sendo este o sistema desenvolvido para a detecção, reconhecimento e classificação dos sinais de trânsito. Mais à direita na figura encontra-se localizada a alimentação do sistema, i.e. as baterias (castanho). A imagem d) da figura anterior mostra o interior do veículo onde se pode visualizar o computador portátil, que recebe as imagens da webcam e a informação proveniente da CMUcam3, relativa ao sinal que foi detectado. As condições do teste são descritas no final do anexo 2.

Numa situação de aplicação definitiva ou com uma vertente comercial, sem dúvida que o sistema sofreria alterações de aparência.

Durante os testes reais foram retirados os seguintes valores:

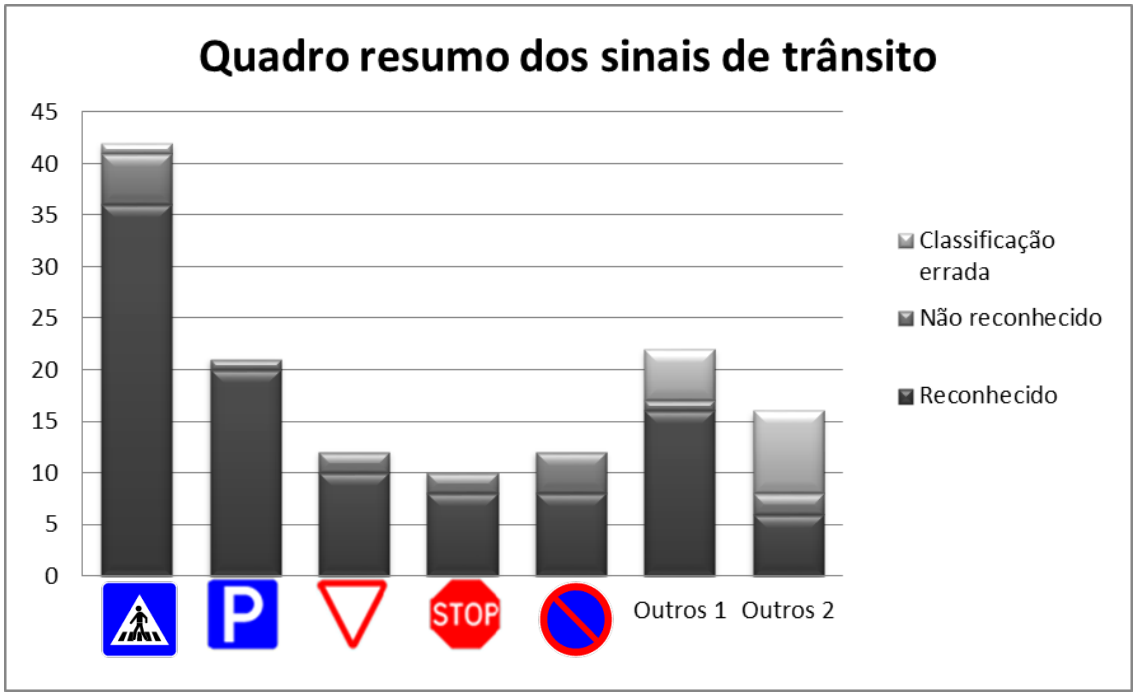


Figura 30 - Quadro resumo dos sinais detectados

No anexo 2 encontram-se alguns dos sinais correctamente classificados e sinais que não foram detectados ou erradamente detectados.

Calculando a percentagem de sinais correctamente detectados, reconhecidos e classificados, obteve-se um valor de 81,9%, segundo os procedimentos de cálculo de Bascón *et al.* (2010) e Paulo e Correia (2000), dado por:

$$\text{Reconhecimento} = \frac{\text{Reconhecidos}}{\text{Analisados}} \tag{15}$$

onde:

Reconhecimento = Percentagem de sinais reconhecidos.

Reconhecidos = Número de sinais correctamente reconhecidos.

Analisados = Número de sinais analisados.

Este valor deve ser considerado, caso o sistema se destine, por exemplo, ao inventário de sinais de trânsito. O cálculo que se segue foi introduzido para se ter uma noção diferente, e deve ser tido em conta quando o sistema não permitir falhas, como por exemplo, num veículo autónomo onde a forma de cálculo deve contemplar que uma classificação errada deve ter mais peso do que um sinal reconhecido e que estes devem ser subtraídos aos sinais que são

reconhecidos com sucesso, desde modo de pensar surgiu então a forma de cálculo que se segue:

$$\text{Reconhecimento} = \frac{\text{Reconhecidos-Não\_reconhecido-2*Classificação\_errada}}{\text{Analisados}} \quad (16)$$

onde:

Reconhecimento = Percentagem de sinais reconhecidos.

Reconhecidos = Número de sinais correctamente reconhecidos.

Não_reconhecido = Número de sinais não reconhecidos.

$\text{Classificação\_errada}$ = Número de sinais analisados que obtiveram uma classificação errada.

Analisados = Número de sinais analisados.

Foi adoptada esta forma de cálculo pois, na perspectiva desta investigação, um sinal reconhecido com valor errado é de imenso perigo, pelo que um valor não reconhecido também deve ser enumerado como negativo. Isto supondo que a aplicação deste sistema se destina, por exemplo, a um carro autónomo. Caso seja aplicado a um inventário de sinais na estrada deve-se considerar os 81,9% referidos anteriormente, até porque na nova forma de cálculo encontram-se sinais visivelmente degradados e marginalizados (visivelmente em más condições). O valor de reconhecimento total foi de 15,3%, o que denota a necessidade de futuras melhorias do sistema para uma implementação real.

6.4. Nota conclusiva

A construção do algoritmo para o reconhecimento de sinais de trânsito, tendo em conta que se destina a um sistema embutido, fez com que se pudesse atingir resultados aceitáveis, quer na perspectiva de tempo de processamento, quer na perspectiva de resultados. Como ponto negativo, salienta-se a não detecção das formas quando estas se apresentam incompletas ou uma amostragem deficiente aquando de ruído em sinais que tenham o pictograma branco.

Capítulo 7 - Conclusões

Conclusão

Neste capítulo final são descritas as observações finais, conclusões e técnicas que devem ser testadas de forma a tentar melhorar o sistema desenvolvido.

Como conclusão retêm-se que o sistema embutido desenvolvido é suficiente para a detecção de sinalética. O sensor de imagem, embora com pouca definição, mostrou-se à altura das exigências e o microcontrolador, pautado por recursos reduzidos, também demonstrou boas capacidades de realizar um processamento rápido.

O *software* desenvolvido superou as expectativas visto que, recorrendo a algoritmos simples, conseguiu realizar as tarefas propostas, tendo sido atingida uma taxa de reconhecimento de 81,9%. O valor significativo da taxa de reconhecimento decorre da simbiose entre as características do *software*, pensado e construído de raiz, para não só satisfazer os objectivos, mas também tendo em conta as capacidades/limitações do *hardware*, sendo nestes factores que reside o triunfo deste trabalho.

Ficou demonstrado que um sistema deste tipo é suficiente para obter informação complementar a outros dispositivos de navegação, nomeadamente o GPS, assegurando desta forma uma actualização em tempo real e uma redundância de irrefutável importância.

Perspectivas de Trabalho Futuro

Embora o sistema embutido seja adequado à detecção de sinaléticas, a quantidade de informação existente, nomeadamente nos pictogramas, torna notório que o trabalho no reconhecimento de sinalética é algo a evoluir no futuro, já que o presente sistema possui pouco espaço de armazenamento, não podendo guardar muitos dados que providenciassem a sua correcta detecção.

Este tipo de sistema pode funcionar complementarmente a outros dispositivos de navegação com o intuito de aumentar a redundância destes e proporcionando uma actualização em tempo real, todavia, uma disponibilidade de um controlador de velocidade superior seria um contributo para o aumento da taxa de actualização.

O sensor CMOS mostrou-se insuficiente para satisfazer todas as necessidades, sendo necessário um sensor que consiga ter não só mais resolução como também preparado para o ambiente exterior, nomeadamente no que toca a variações de luminosidade e condições de luminosidade reduzida. A execução de testes com lentes com mais abertura poderia revelar

um bom contributo, pois poderia ser adquirida mais informação, visto que os sinais de trânsito não só aparecem à direita mas também à esquerda.

O algoritmo de detecção de forma deve ser evoluído de modo a permitir que se detecte uma forma mesmo esta não estando completa.

Embora o espaço de cores HSV tenha sido de imenso contributo, na prática tem alguns aspectos menos positivos que embora de menor importância poderão ser melhorados caso se utilize um espaço que abstraia por completo a luminosidade, e para tal, explorar o espaço de cores HSL (*Hue*, *Saturation and Luminance*).

Bibliografia

Aoyagi, Y., Asakura, T., "*A study on traffic sign recognition in scene image using genetic algorithms and neural networks*", Proceeding of the 22nd IEEE International Conference on Industrial Electronics, Control and Instrumentation, vol. 3, Taipeh, Taiwan, 1996.

Bascón, S.M., Rodríguez, J.A., Arroyo, S.L., Caballero, A.F. e Ferreras, F.L., "*An optimization on pictogram identification for the road-sign recognition task using SVMs*", Computer Vision and Image Understanding, Vol. 114, Issue 3, March 2010, pp. 373-383.
Boughen, Nicholas, "*Lightwave 3D 7.5 Lighting*", Wordware Publishing Inc., March 2010.

Bueno, R. Vicen, Gil-Pita, R., Rosa-Zurera, M., Utrilla-Manso, M., López-Ferreras, F., "*Multilayer perceptrons applied to traffic sign recognition tasks*", Proceedings of the 8th International Work-Conference on Artificial Neural Networks, Barcelona, Spain, 2005.

Carlson, J., St. Onge, S., "*Traffic Sign Detection Senior Project Final Report*", Bradley University, May 2008.

CodeSourcery, "*Sourcery C++ Lite: Getting Starded*", CodeSourcery Inc., 2009.

Cyganek, B., "*Circular road signs recognition with soft classifiers*", Integrated Computer-Aided Engineering 14, vol. 4, August 2007.

Cygwin, "*Cygwin User's Guide*", Red Hat Inc., 2010.

de la Escalera, A., Armingol, J.M., Mata, M., "*Traffic sign recognition and analysis for intelligent vehicles*", Image and Vision Computing 21, March 2003.

de la Escalera, A., Armingol, J.M., Pastor, J.M., Rodríguez, F.J., "*Visual sign information extraction and identification by deformable models for intelligent vehicles*", IEEE Transactions on Intelligent Transportation Sytems 15, vol. 2, June 2004.

de la Escalera, A., Moreno, L.E., Puente, E.A., Salichs, M.A., "*Neural traffic sign recognition for autonomous vehicles*", 20th International Conference on Industrial, Electronics, Control and Instrumentation, IEEE, IECON'94, vol. 2, New York, September 1994.

Diário da República, "Decreto Regulamentar 22-A/98", I Série-B, vol. 227, October 1998.

Douville, P., "*Real-time classification of traffic signs*", Real-Time Imaging 6, 2000.

Frias-Martinez, E., Sanchez, A., Valez, J., "*Support vector machines versus multilayer perceptrons for efficient off-line signature recognition*", Engineering Applications of Artificial Intelligence 19, vol. 6, March 2006.

Hibi, T., "*Vision based extraction and recognition of road sign region from natural color image, by using HSL and coordinates transformation*", Proceedings of the 29th International Symposium Automotive Technology and Automation, Florence, Italy, 1996.

Hsu, S. H., Huang, C. L., "*Road sign detection and recognition using matching pursuit method*" Image and Vision Computing 19, December 2000.

Intel, "*Open Source Computer Vision Library Reference Manual*", Intel Corporation, 2001.

Jones, P. , Viola, M., "*Rapid Object Detection using a Boosted Cascade of Simple Features*", IEEE, 2001.

Kang, D., Griswold, N., Kehtarnavaz, N., "*An invariant traffic sign recognition system based on sequential color processing and geometrical transformation*", Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation, Dallas, Texas, April 1994.

Kraiss, K., "*LTI Image Processing Library Developer's Guide*", Rheinisch-Westfalische Technische Hochschule Aachen, June 2004.

Krebel, U., Lindner, F., Wohler, C., Linz, A., "*Hypothesis verification baed on classification at unequal error rates*", Proceedings of the IEEE Artificial Neural Networks, September 1999.

Lienert, A., "*Tech Watch: Opel Eye Warns Drivers About Speed Limits*", Edmunds Inc., June 2008.

Lorsakul, A., Suthakorn, J., "*Traffic sign recognition using neural network on OpenCV: Toward intelligent vehicle/driver assistance system*", Mahidol University, 2007.

Luo, R., Potlapalli, H., Hislop, D., "*Neural network based landmark recognition for robot navigation*", IEEE Transactions on Intelligent Transportation Systems 7, vol. 3, November 1992.

Maldonado, S., Lafuente, S., Gil, P., Gomez, H., Lopez, F., "*Road-sign detection and recognition based on support vector machines*", IEEE Transaction on Intelligent Transportation Systems 8, June 2007.

Mariano, S., "*Sistemas de decisão óptima em coordenação hidrotérmica para planeamento operacional*", Universidade da Beira Interior, 2000.

MathWorks, "*Introduction to MATLAB*", MathWorks, 2008.

Miura, J., Kanda, T., Shirai, Y., "*An active vision system for real-time traffic sign recognition*", Proceddings of the IEEE Intelligent Transportation Systems. October 2000.

General Motors, Deacon, C., "*New "Opel-Eye" Technology Can Read Signs*", June 2008.

NXP, "*NXP LPC2104_2105_2106_7*", NXP, 2008.

Omnivision, "*OV6620 single-chip CMOS CIF color digital camera*", Omnivision, 2000.

Paclick, P., Novovicova, J., Duin, R., "*Building road-sign classifiers using a trainable similarity measure*", IEEE Transactions on Intelligent Transportation Systems 7, vol. 3, September 2006.

Paulo, C. F., Correia, P. L., "*Automatic detection and classification of traffic signs*", Instituto de Telecomunicações, Instituto Superior Técnico, June 2000.

Perez, E., Javidi, B., "*Nonlinear distortion-tolerant filters for detection of road signs in background noise*", IEEE Transactions on Vehicular Technology 51, vol. 3, May 2002.

Polastre, J., Szewczyk, R. , Culler, D., "*Telos: Enabling Ultra-Low Power Wireless Research*", Spots, April 2005.

Quantum Leaps, "*Building Bare-Metal ARM systems with GNU*", Quantum Leaps, August 2007.

Rowe, A., Goode, A., Carnegie Mellon University, "*CMUcam3 software installation guide v0.1 for the CMUcam3 embedded vision sensor*", Illah Nourbakhsh, April 2006.

Rowe, A., Goode, A., Goel, D., Nourbakhsh, I., "*CMUcam3: An Open Programmable Embedded Vision Sensor*", Carnegie Mellon University, May 2007.

Ryzhyk, L., "*The ARM Architecture*", June 2006.

Shojania, H., "*Real-time Traffic Sign Detection*", October 2003.

Siegmann, P., López, R., Gil, P., Lafuente, S., Maldonado, S., "*Fundamentals in luminance and retroreflectivity measurement of vertical traffic signs using a color digital camera*", IEEE Transactions on Instrumentation and Measurement 57, March 2008.

Song, H, Bolouki, S., Yen, A., "*Detection of Stop Signs*", Stanford University, 2008.

van Dimitri, Heesch, "*User Manual for doxygen 1.5.3*", June 2007.

Anexos

Anexo 1

Introdução

Neste anexo encontram-se os sinais de trânsito importados do Decreto-Regulamentar n.º22-A/98, de 1 de Outubro, que serviram de suporte para a construção da base de dados.

8.1.1. Sinais de trânsito

100


200


300*


301


302


303


304


305


306


307

308

309


310


311


312


313


314


315


316


317


318


319


320


321


322


323


324


325


326


327


328


329


330


331


332


333


334


335


336


337

338

339

340

341

342

343

344

345

346

347

348

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

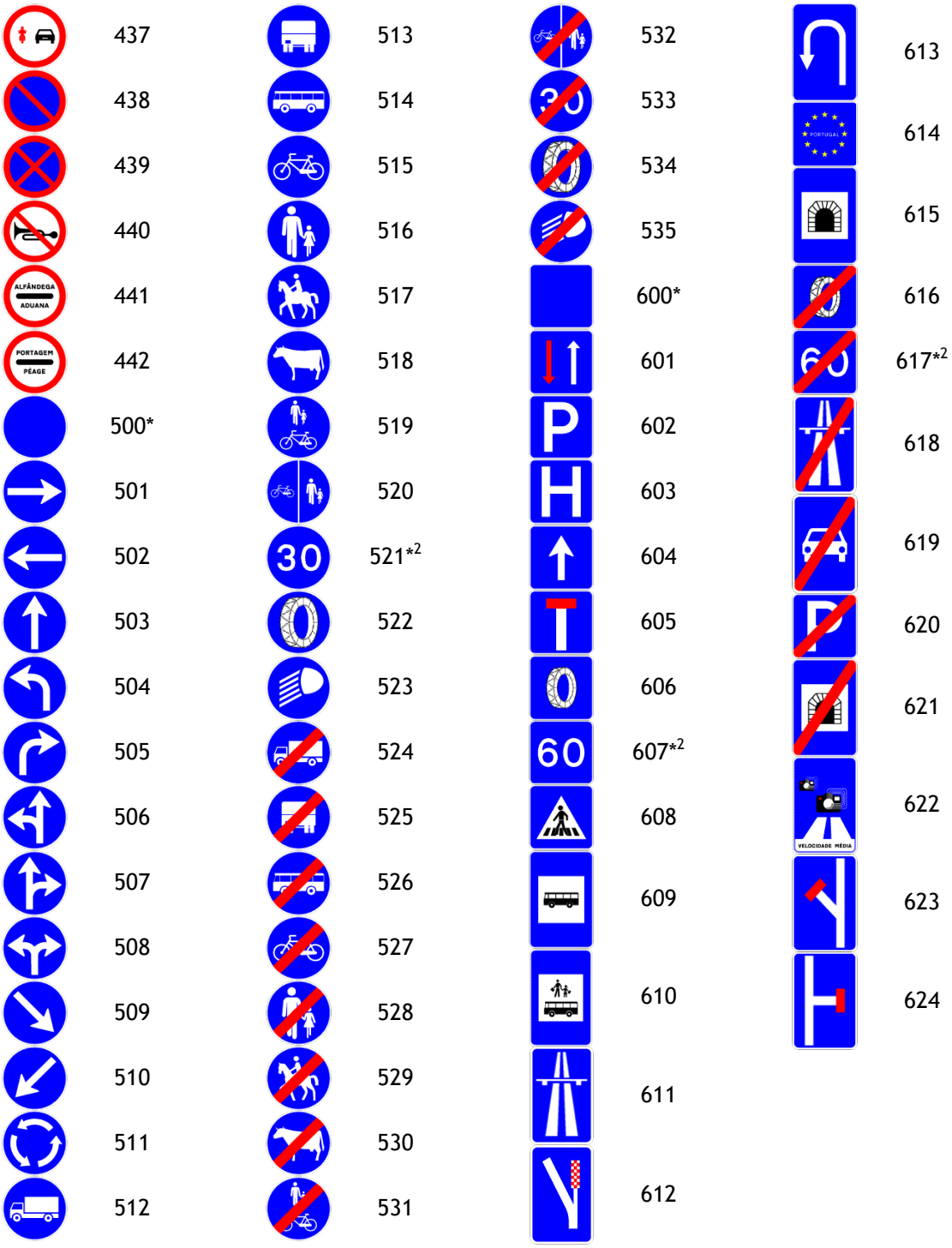
432

433

434*2

435

436



*- Sinais base.
*2 - Sinais que podem conter outra informação similar

8.1.2. Símbolos

Os símbolos são complementos dos sinais base, que devido a surgirem com uma frequência reduzida na estrada, são armazenados em memória SD. A sua codificação é feita do seguinte modo:

$$símbolo = sinal_{base} + \text{"s"} + código_{símbolo} + \text{"p"} + probabilidade$$

	1		24		45		68
	2		25		46		69
	3		26		47		70
	4		27		48		71
	5		28		49		72
	6		29		50		73
	7		30		51		74
	8		31		52		75
	9		32		53		76
	10		33		54		77
	11		34		55		78
	12		35		56		79
	13		36		57		80
	14		37		58		81
	15		38		59		82
	16		39		60		83
	17		40		61		84
	18		41		62		85
	19		42		63		86
	20		43		64		87
	21		44		65		88
	22				66		89
	23				67		90

 91

 92

 93

 94

 95

 96

 97

 98

 99

 100

 101

 102

 103

 104

 105

 106

 107

 108

 109

Anexo 2

Introdução

Exemplo de sinais que mesmo ao não se encontrarem nas condições perfeitas, foi conseguido o seu reconhecimento, assim como de exemplos de sinais que não foram reconhecidos.

8.2.1. Sinais de trânsito reconhecidos

Nesta secção encontram-se seleccionados alguns sinais de trânsito que se destacaram no processo de detecção, reconhecimento e classificação por parte do sistema desenvolvido.

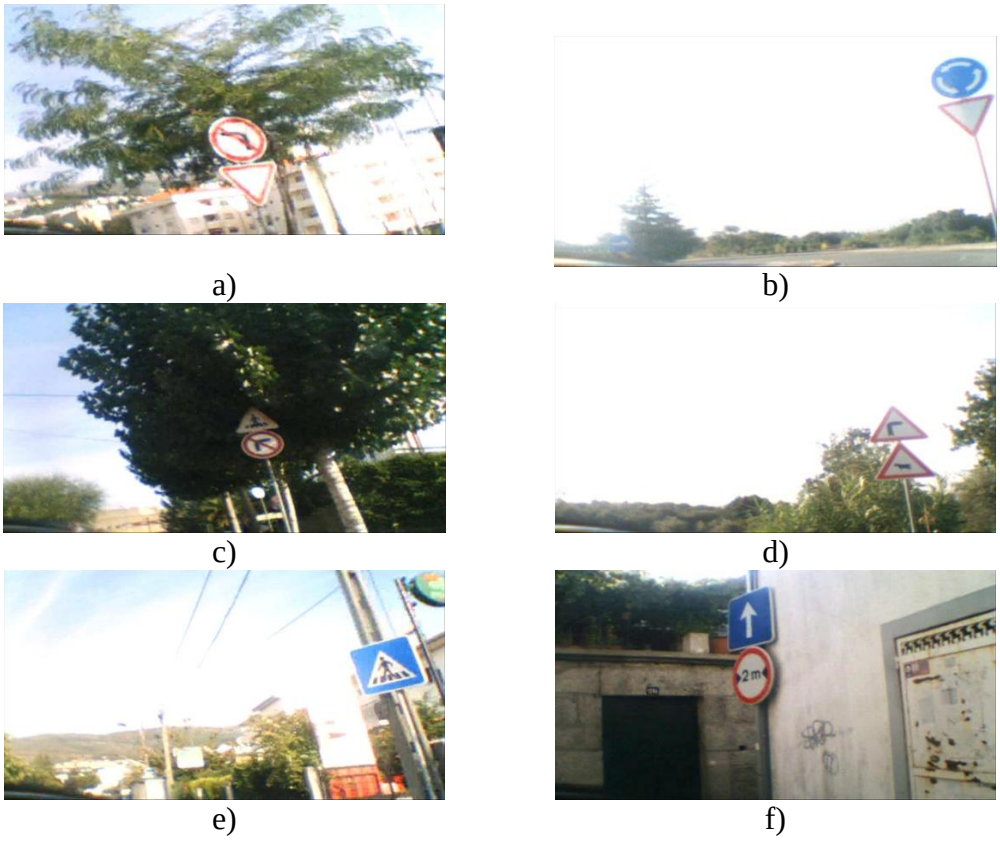


Figura 31 - Exemplo de sinais reconhecidos

Em exemplo foram escolhidas as imagens na figura anterior, pois demonstram uma panóplia de sucesso por parte do sistema desenvolvido. Particularmente, as imagens a), b), d) e e) são caracterizadas por um fundo de imensa intensidade, cuja resolução se deveu ao espaço de cores utilizado (HSV). A imagem c) foi seleccionada por apenas conseguir detectar o sinal inferior (proibido virar à direita) e fazer um contraste de sinais que consegue reconhecer, isto porque o sinal de passagem de peões está visivelmente degradado. Nas imagens e) e f) denota-se que os sinais não estão correctamente inclinados para a câmara, fazendo com que a sua resolução fique a cargo do algoritmo "puxa orelhas".

8.2.2. Sinais de trânsito erradamente reconhecidos

Nesta secção encontram-se os sinais de trânsito que foram erroneamente classificados, i.e. classificados como falsos positivos.

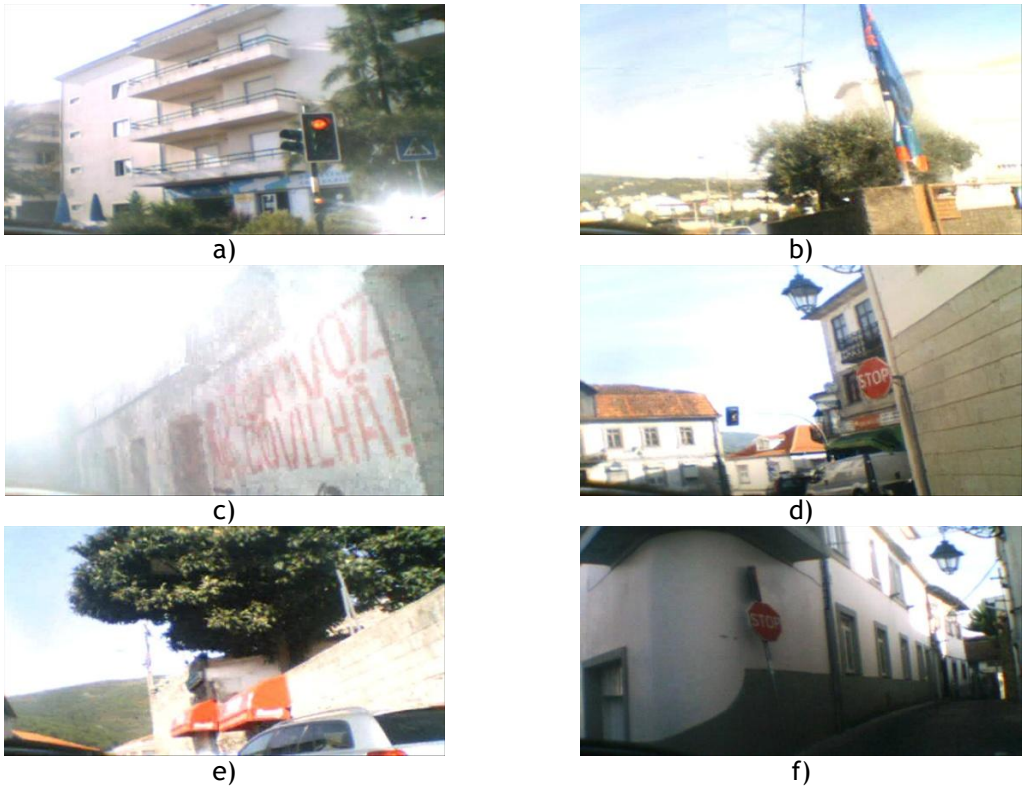


Figura 32 - Exemplo de sinais erradamente reconhecidos.

Na figura anterior foram seleccionadas algumas amostras de não sinais que foram caracterizados como sendo. Na imagem a), c) e e) (sinal 402 referente ao anexo anterior), foi caracterizado como uma via de sentido proibido.

Na imagem b) como uma estrada sem saída (sinal 605 do anexo 1) e na imagem d) a publicidade da loja foi caracterizada como uma via de sentido proibido (sinal 402 referente ao anexo anterior).

Na figura f), embora tenha procedido ao reconhecimento do sinal de STOP, foi considerado como um forma errónea de caracterizar pois o sinal era proveniente de outra rua. Numa aplicação de condução autónoma, o carro iria parar sem motivo para tal.

8.2.3. Sinais de trânsito não reconhecidos

Em seguida segue a lista de alguns sinais que não foram reconhecidos.

Figura 33 - Exemplo de imagens não reconhecidas

O conjunto de imagens da figura anterior foi seleccionado por não ter reconhecido os sinais de trânsito. As razões pela qual as imagens a), b), d) e f) não foram reconhecidas, deve-se ao facto da enorme luminosidade solar que incidia no sensor de imagem.

Na imagens c) e e) deveu-se ao facto de o sinal estar parcialmente incompleto e por isso não ser possível reconhecer a forma. Na imagem g), a vandalização do sinal foi o motivo pelo qual nem a forma foi conseguida.

8.2.4. Condições do teste prático

O teste prático foi realizado entre os dias 20 e 22 de Setembro de 2010, em períodos de dia alternados, de modo a realizar testes diurnos e nocturnos em diversas zonas da Covilhã e arredores. O ambiente de testes complementou zona urbana e fora da localidade, onde as velocidades do veículo foram no máximo de 50 km/h em zona urbana e de 70 km/h fora da

localidade. Devido às condições de visibilidade reduzida e a escassez de sinais na mesma, não foram realizados testes em auto-estrada. Outro factor que não comprovado residiu na existência de condições climatéricas adversas o que traria um valor acrescentado à fiabilidade deste teste prático, uma vez que no intervalo de dias de teste, as condições climatéricas foram de sol. As estradas percorridas encontram-se delimitadas entre o Pelourinho da Covilhã e o Refúgio-Covilhã.