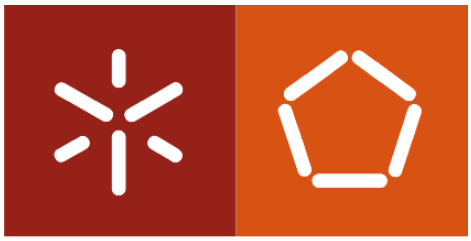


Universidade do Minho

Mestrado em Engenharia Informática

Perfil de Computação Gráfica

Unidade Curricular de Visão por Computador

Análise temporal do Spacial Power Spectra de diferentes imagens

Docente: Doutora Cristina Peixoto Santos

Alunos: Carlos de Castro - ID5577

Serafim Pinto - PG28506

Resumo

No âmbito da Unidade Curricular de Visão por Computador, do perfil de Computação Gráfica do Mestrado em Engenharia Informática, este relatório pretende apresentar o projeto de Análise Temporal, adotando o algoritmo que nos foi fornecido pelo artigo “A Time-analysis of the Spatial Power Spectra indicates the proximity and complexity of the surrounding environment”, bem como as decisões que foram tomadas ao longo do projeto e as dificuldades sentidas. O principal objetivo foi fazer uma implementação em C++ recorrendo à biblioteca OpenCV, seguindo o algoritmo referido, através da análise de diferentes sequências de imagens.

Índice

Introdução

O presente documento serve de suporte à realização de todo o trabalho desenvolvido ao longo do semestre, da Unidade Curricular de Visão por Computador.

O desenvolvimento deste projeto tem como ponto de partida um artigo científico com o título “A Time-analysis of the Spatial Power Spectra indicates the proximity and complexity of the surrounding environment” publicado por duas investigadoras da Universidade do Minho, Ana Carolina Silva e Cristina Peixoto Santos, sendo a última autora a docente desta Unidade Curricular.

A percepção visual tem uma extrema importância no âmbito da robótica, nomeadamente no controlo de tarefas complexas, como é o caso da navegação autónoma em ambientes não estruturados ou por exemplo na prevenção de colisões. Infelizmente a visão é uma tarefa extremamente complexa.

Uma abordagem bem fundamentada que aposta na resolução dos problemas da visão por computador é baseada nas estatísticas de imagem e provém da área da biologia.

Quando duas imagens de espaços naturais próximos são comparadas, pode afirmar-se que a sua intensidade também é próxima. Uma análise mais aprofundada nesta área científica provou a necessidade da inclusão da análise temporal à análise estatística das imagens. De facto, quer o observador, quer os objetos ou cenas observadas possuem movimento ao longo do eixo temporal.

Posto isto, pode dizer-se que o principal objetivo deste trabalho passa por implementar o algoritmo já mencionado, e essencialmente tentar detetar a aproximação de objetos, para no futuro implementar esta solução num ambiente de robótico e ser possível que estes desviem a sua trajetória desses mesmos objetos de forma a evitar colisões.

Motivação

De acordo com a nova metodologia desenvolvida pelas autoras do artigo, baseada numa computação em tempo real do “Image Power Spectra”, pretendeu-se implementar uma solução com recurso à linguagem de programação C++, recorrendo à biblioteca de software OpenCV[1].

A implementação desta solução tem como principais objetivos a obtenção do movimento de objetos num determinado ambiente, a determinação da complexidade do ambiente envolvente e a segurança da trajetória a percorrer pelo robô.

[1] http://opencv.org

Abordagem ao problema

Para atingir os objetivos propostos, foi desenvolvido um algoritmo, no qual em primeiro lugar foi obtida a imagem da câmera computador através da criação de um objeto da classe VideoCapture:

```
VideoCapture stream(0).
```

stream 0 - obtêm imagem da câmera

stream 1 - obtêm imagem de um *path*, podendo este ser um vídeo ou uma imagem.

De forma a efetuar os primeiros testes à aplicação, optou-se por utilizar a câmera do computador, no entanto numa fase final do desenvolvimento o *stream* de vídeo da câmera foi substituído por vídeos com sequências de imagens pré gravados com contextos de ambientes específicos para o efeito.

Para que o passo seguinte fosse calculado com sucesso, procedeu-se à correção da dimensão da imagem capturada de forma a que a mesma tenha sempre um formato quadrado, ou seja, a sua largura seja sempre igual à sua altura. Desta forma, são adicionadas nas respetivas extremidades da imagem barras pretas de forma a compensar a dimensão com menor valor.

A função desenvolvida para esta funcionalidade tem o nome:

```
turnIntoSquareImage
```

A função acima, recebe uma *frame* como parâmetro de entrada, verifica o tamanho da matriz da imagem e caso esta esteja descompensada na horizontal ou na vertical, são adiciondas estas barras pretas respetivamente, como podemos ver pela imagem acima. Esta passo é aplicado mesmo antes da implementação descrita a seguir.

De seguida é invocada a função `discreteFourierTransform` que aplica a transformada de *Fourier* a uma imagem e devolve um objeto do tipo `Mat` (Matriz) com a magnitude da imagem. Esta função resolve a seguinte fórmula apresentada no artigo:

$$\mathrm{FT}_t(f_x, f_y) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I_t(x,y) e^{-i\frac{2\pi}{N}(f_x x + f_y y)}$$

É a partir desta matriz que a função devolve, que todo o algoritmo ganha forma. A partir deste momento teremos que ter em atenção vários passos, e com bastante cuidado ao aplicar os algoritmos à imagem. Foi a partir daqui que o grupo teve mais dificuldade no desenrolar do projeto, pois não se conseguiu perceber de uma forma direta o que era necessário fazer, e para isso teve que se contactar a equipa docente, e depois de alguma ajuda, através de algum código desenvolvido em *matlab* fornecido, conseguimos finalmente avançar, ainda assim com alguns problemas, como referido mais adiante.

Posto isto, de seguida é apresentada a função `powerspectra` que é onde está implementado o resto do algoritmo. Este método recebe como parâmetro a matriz mangitude (resultado da função `discreteFourierTransform`) e percorre cada pixel dessa matriz, elevando-a ao quadrado.

$$\mathrm{PS}_t(f_x, f_y) = |\mathrm{FT}_t(f_x, f_y)|^2$$

Entretanto, são criados dois vetores (X e Y) relativos à frequência da imagem, cuja dimensão vai de -M/2 até (M/2 - 1), em que M equivale ao tamanho da imagem.

A estes vetores é aplicada a função `meshgridTest`, que foi adaptada do *matlab* para o OpenCV:

```
meshgridTest(cv::Range(-imagesize/2, imagesize/(2-1)),
   cv::Range(-imagesize/2, imagesize/(2-1)), x, y);
```

Esta função replica os vetores recebidos como parâmetro para produzir uma matriz completa nas dimensões X e Y. O tamanho da matriz de saída é determinado pelo comprimento dos vectores de entrada. Para vectores de entrada com comprimento M e N respectivamente, a matriz de saída terá N linhas e M colunas.

Primeiramente foi testado em *matlab* com alguns exemplos, para verificar se o que estava a ser desenvolvido no OpenCV era o correto. Um exemplo de aplicação prática desta função seria o seguinte:

```
[X,Y] = meshgrid(1:3,10:14)
X =
     1     2     3
     1     2     3
     1     2     3
     1     2     3
     1     2     3
Y =
    10    10    10
    11    11    11
    12    12    12
    13    13    13
    14    14    14
```

Após esta operação foi necessário converter as coordenadas dos dois vetores do sistema cartesiano (eixo dos X e Y) para o sistema polar. Para tal, foi utilizada a função já existente do OpenCV:

```
cv::cartToPolar(X, Y, rho, theta);
```

Como output esta função devolve o ângulo *theta* que neste caso será desprezado pois não é necessário neste algoritmo, e a *rho* que neste caso indica a magnitude. Quer *theta*, quer *rho* são variáveis do tipo `Mat`, ou seja, são matrizes.

De forma a estabelecer uma comparação entre os valores da matriz *rho* e a matriz resultante da função `powerspectra`, procedeu-se ao arredondamento do valor de cada posição dessa matriz para um número inteiro da seguinte forma, em que se aplicou o Round a cada posição da matriz:

```
for (int i = 0; i < rho.rows; i++)
for (int j = 0; j < rho.cols; j++)
     rho.at<float>(i, j) = cvRound(rho.at<float>(i, j));
```

De seguida, para cada posição unitária, de 1 até metade do tamanho da imagem (M/2), foi percorrida a matriz *rho*, já com todas as suas posições com números inteiros, no sentido de encontrar os valores inteiros sequenciais em *rho* de 1 até metade do tamanho da imagem (M/2). Neste caso, na mesma posição da matriz devolvida na função `powerspectra`, esse valor será guardado num vetor auxiliar:

```
valoresDFT.push_back(in.at<float>(k, j));
```

Ou seja, para cada magnitude da matriz *rho*, foram guardadas as posições e posteriormente foi-se à matriz da DFT anteriormente calculada ao quadrado, e calcularam-se as médias para todos os valores desejados.

Para cada posição do ciclo inicial foi calculada uma média com os valores do *array* `valoresDFT`. No final desta operação obteve-se um *array* `medias` que permitiu juntamente com o *array* `R`, cujo conteúdo é igual a valores de 1 até metade do tamanho da imagem (M/2) calcular o valor de alfa através da função `fitline`.

Este foi também um passo bastante importante, pois é aqui que este algoritmo começa a fazer sentido, de certa forma, isto é, agora que existe o eixo dos X (as médias) e o eixo dos Y (R de 1 até M/2) pode-se, para cada *frame,* calcular o declive da recta formada por estes pontos e assim obter o alfa. Será esta variável que irá indicar, se o algoritmo está funcionar corretamente e se realmente se consegue detetar a aproximação de objetos com eficácia.

Voltando outra vez à `fitline`, que é a equivalente à função `polyfit` do *matlab*:

```
fitLine(points,line,CV_DIST_L2,0,0.01,0.01);
```

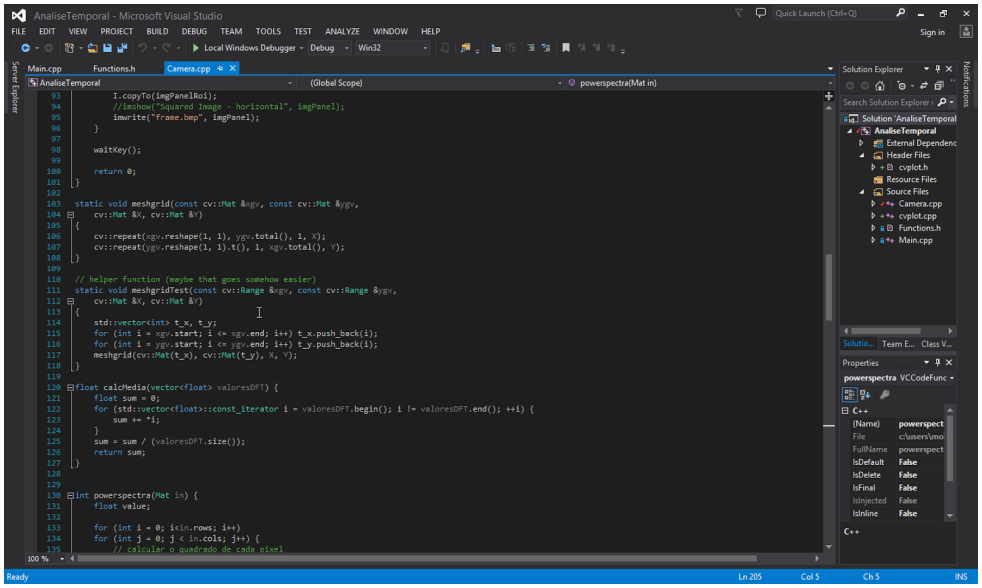
O output desta função recebe como entrada um conjunto de pontos, referido anteriormente e mais algumas configurações, e devolve a reta aproximada para esses mesmos pontos. Através desta reta, consegue facilmente obter-se o seu declive que é o tão desejado `alpha`.

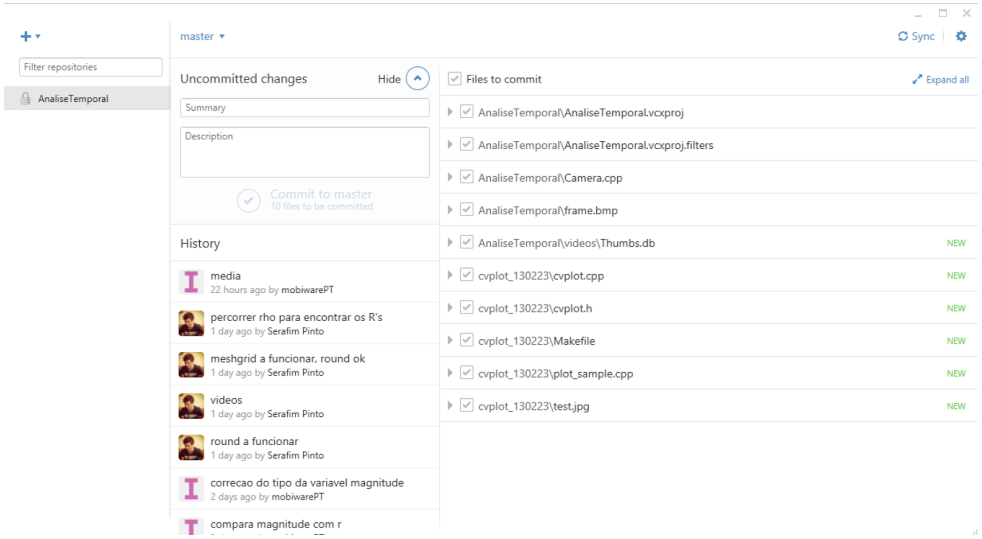
Desta forma, a solução para o problema está quase encontrada, pois é através desta variável que se pode avaliar a análise às sequências de imagens (*frames*).

Chegado este ponto do trabalho, é altura de analisar diferentes sequências de imagens, comparar e retirar conclusões, como apresentado de seguida.

Solução e Validação

A solução apresentada foi desenvolvida no IDE[2] Microsoft Visual Studio 2013, no Sistema Operativo Windows 8 64 bits, sendo que como o desenvolvimento do código foi partilhado entre os dois elementos do grupo, foi utilizada a plataforma GitHub[3] para o controlo de versões do código gerado.





[2] Ambiente integrado para desenvolvimento de software
[3] https://github.com

Numa fase inicial, apenas para comprovar o funcionamento sem erros da solução, utilizou-se a câmera do computador para correr a aplicação, carregando os vídeos e fazer uma análise.

Numa fase posterior a câmera do computador foi substituída por vídeos pré concebidos para o efeito, que foram fornecidos pela equipa docente, com o objetivo de testar o comportamento de um robô numa situação de navegação simulada. Convém mencionar aqui, que por vários motivos o grupo não conseguiu incorporar a nossa implementação no ambiente do Webots, onde poderíamos simular o comportamento de um robô. Após algumas tentativas, não conseguimos configurar o Webots e assim poder incorporar a nossa aplicação. Desta forma o que fizemos foi, para cada vídeo analisar o alfa *frame-by-frame*, e criar alguns gráficos para perceber melhor o comportamento desta variável.

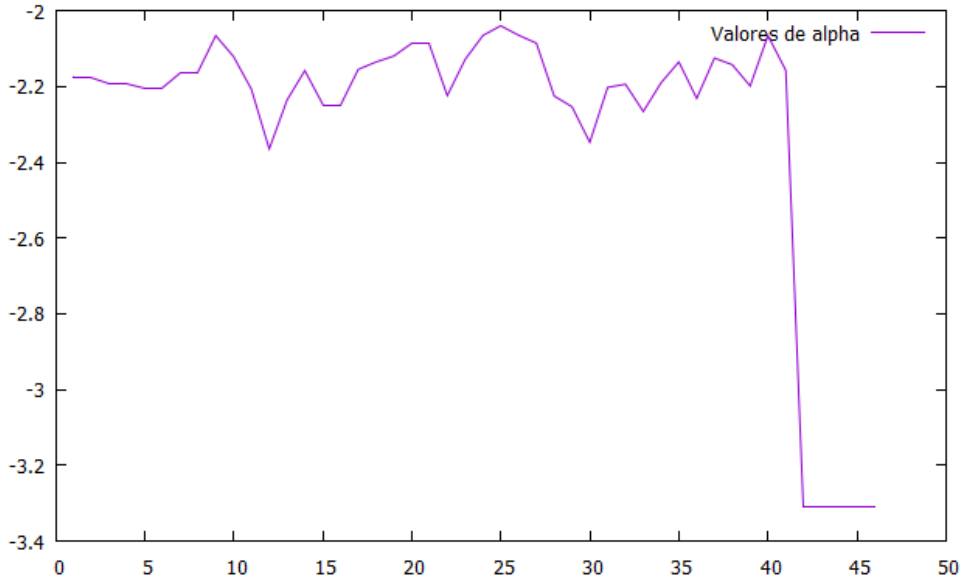
Para esta situação foi possível obter valores de alfa aceitáveis, sendo que os resultados foram guardados automcaticamente num ficheiro de texto e posteriormente carregados e gerados pela ferramenta *gnuplot*[4] para transformação e apresentação da informação em formato gráfico. Procederemos então à análise de resultados.

[4] http://www.gnuplot.info

Para o vídeo approaching_lv_40ms_translate_approach.avi (aproximação de um quadrado preto), os valores de alfa calculados são os seguintes:

1 -2.175022	24 -2.0655525
2 -2.175023	25 -2.0400926
3 -2.193684	26 -2.0649127
4 -2.193685	27 -2.0857328
5 -2.205866	28 -2.225429
6 -2.205867	29 -2.2536530
7 -2.162198	30 -2.3475431
8 -2.162199	31 -2.2025432
9 -2.0660710	32 -2.1940733
10 -2.1213711	33 -2.2661534
11 -2.2073512	34 -2.1905435
12 -2.36513	35 -2.1354736
13 -2.2361714	36 -2.2317237
14 -2.1583415	37 -2.1252590
15 -2.2504116	38 -2.14284221
16 -2.2504117	39 -2.1989222
17 -2.1551318	40 -2.06582223
18 -2.1354219	41 -2.15759224
19 -2.1198120	42 -3.30951225
20 -2.0870721	43 -3.30951226
21 -2.0870722	44 -3.30951227
22 -2.2243323	45 -3.30951228
23 -2.1286924	46 -3.30951

O gráfico gerado para os valores apresentados acima é o seguinte:



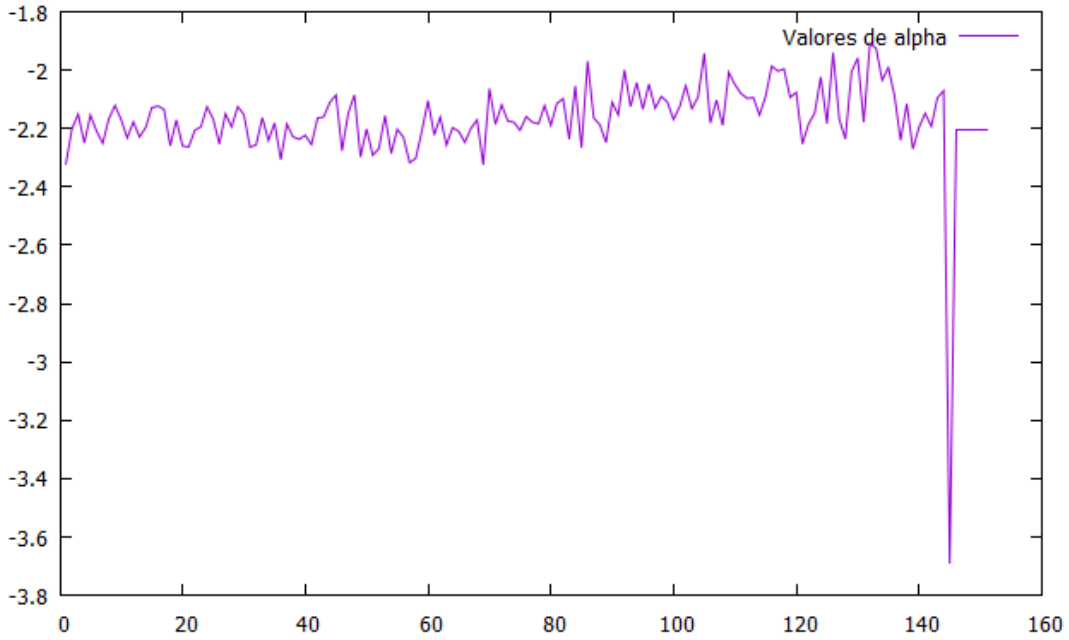
Após uma análise aos valores de alfa obtidos, pode comprovar-se, tal como esperado, que no momento em que o quadrado preto do vídeo aumenta de tamanho, ou seja, o objeto se aproxima da câmera, há um disparo considerável no valor de alfa, pelo que numa implementação num robô, deveria ser enviado um *spike* ao mesmo para que este se desviasse do obstáculo encontrado.

Os valores seguintes apresentam os resultados de alfa para o vídeo 1_bola.avi:

1 -2.32098	36 -2.30631	71 -2.18572	106 -2.18069
2 -2.2001	37 -2.18448	72 -2.11871	107 -2.10292
3 -2.1508	38 -2.2283	73 -2.17389	108 -2.18923
4 -2.24914	39 -2.23661	74 -2.17847	109 -2.00706
5 -2.15388	40 -2.22307	75 -2.20589	110 -2.04909
6 -2.20938	41 -2.25467	76 -2.15913	111 -2.08024
7 -2.25054	42 -2.16431	77 -2.17838	112 -2.09658
8 -2.16648	43 -2.16008	78 -2.18372	113 -2.09396
9 -2.12095	44 -2.11048	79 -2.12145	114 -2.15318

10 -2.16973	45 -2.08519	80 -2.18874	115 -2.09058
11 -2.23203	46 -2.27553	81 -2.11417	116 -1.98581
12 -2.17729	47 -2.14794	82 -2.09741	117 -2.00322
13 -2.22878	48 -2.0853	83 -2.2362	118 -1.99545
14 -2.19606	49 -2.29735	84 -2.05375	119 -2.09317
15 -2.1283	50 -2.20101	85 -2.266	120 -2.07605
16 -2.12284	51 -2.29184	86 -1.96898	121 -2.25276
17 -2.13488	52 -2.26831	87 -2.16333	122 -2.1848
18 -2.25895	53 -2.15562	88 -2.18718	123 -2.1464
19 -2.17047	54 -2.28549	89 -2.24797	124 -2.02326
20 -2.25913	55 -2.20226	90 -2.10929	125 -2.18346
21 -2.26368	56 -2.23018	91 -2.1532	126 -1.93996
22 -2.20577	57 -2.31699	92 -1.99914	127 -2.16773
23 -2.19374	58 -2.30194	93 -2.1247	128 -2.23559
24 -2.12545	59 -2.20888	94 -2.04208	129 -2.00597
25 -2.1681	60 -2.10426	95 -2.1334	130 -1.95791
26 -2.25303	61 -2.22225	96 -2.04737	131 -2.17811
27 -2.14931	62 -2.16025	97 -2.13059	132 -1.90912
28 -2.19427	63 -2.25693	98 -2.09026	133 -1.92496
29 -2.12514	64 -2.19688	99 -2.10898	134 -2.03438
30 -2.15194	65 -2.2093	100 -2.1688	135 -1.99123
31 -2.264	66 -2.24752	101 -2.12502	136 -2.08765
32 -2.25552	67 -2.19904	102 -2.05374	137 -2.24011
33 -2.16279	68 -2.17047	103 -2.13121	138 -2.11477
34 -2.24098	69 -2.32448	104 -2.09161	139 -2.26967
35 -2.18092	70 -2.063	105 -1.94251	140 -2.19548
			141 -2.14815
			142 -2.1927
			143 -2.09475
			144 -2.07051
			145 -3.69157
			146 -2.2059
			147 -2.2059
			148 -2.2059
			149 -2.2059
			150 -2.2059
			151 -2.2059

O gráfico gerado para os valores apresentados acima é o seguinte:

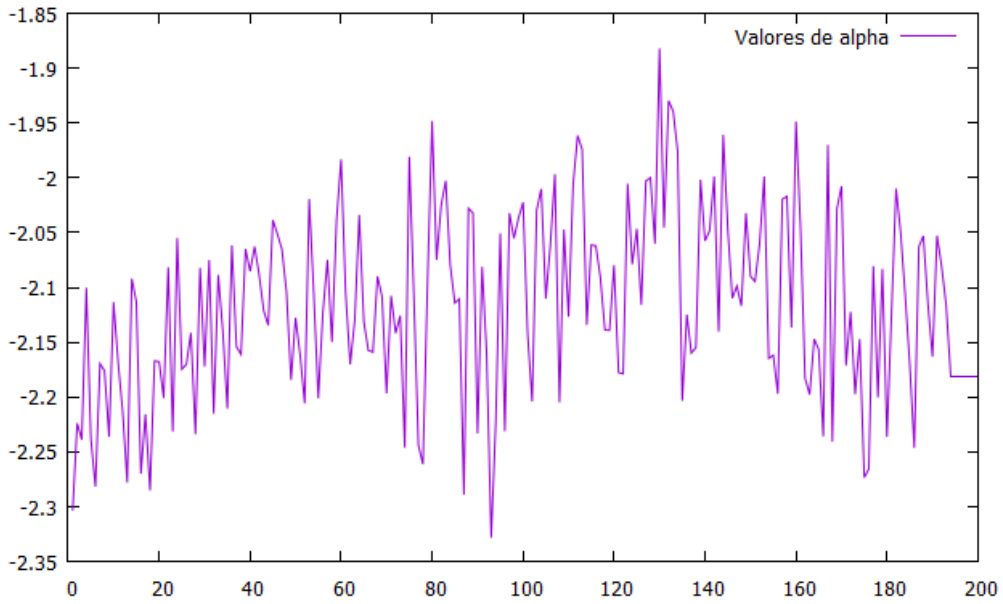


A análise dos valores de alfa produzidos após o processamento deste vídeo é idêntica à análise anterior, sendo que a conclusão retirada é a mesma. Com a aproximação da bola à câmera, o alfa aumenta drasticamente, o que é bom pois significa que deteta uma aproximação.

Os valores seguintes apresentam os resultados de alfa para o vídeo 2_bolas.avi:

1 -2.30294	51 -2.16173	101 -2.14161	151 -2.09451
2 -2.22381	52 -2.20571	102 -2.20408	152 -2.06022
3 -2.23886	53 -2.01936	103 -2.02954	153 -1.9988
4 -2.10036	54 -2.10614	104 -2.01006	154 -2.16459
5 -2.23771	55 -2.20096	105 -2.1103	155 -2.16186
6 -2.28167	56 -2.12337	106 -2.06172	156 -2.19699
7 -2.16955	57 -2.07482	107 -1.99666	157 -2.01948
8 -2.17599	58 -2.14981	108 -2.20489	158 -2.01698
9 -2.23634	59 -2.03884	109 -2.04719	159 -2.1366
10 -2.1134	60 -1.98326	110 -2.12667	160 -1.94871
11 -2.16866	61 -2.10427	111 -2.0063	161 -2.04627
12 -2.21392	62 -2.17017	112 -1.96143	162 -2.18272
13 -2.27797	63 -2.13004	113 -1.97422	163 -2.19758
14 -2.09203	64 -2.03387	114 -2.1342	164 -2.14715
15 -2.11327	65 -2.13116	115 -2.06129	165 -2.15656
16 -2.27004	66 -2.15759	116 -2.06209	166 -2.23586
17 -2.21561	67 -2.1589	117 -2.09073	167 -1.96997
18 -2.28515	68 -2.08994	118 -2.13873	168 -2.24067
19 -2.16709	69 -2.10832	119 -2.13917	169 -2.02829
20 -2.16767	70 -2.19658	120 -2.0798	170 -2.00768
21 -2.20068	71 -2.10757	121 -2.17757	171 -2.17116
22 -2.08161	72 -2.14177	122 -2.17878	172 -2.12222
23 -2.23143	73 -2.12569	123 -2.00539	173 -2.19737
24 -2.05494	74 -2.24616	124 -2.07885	174 -2.14713
25 -2.17452	75 -1.9808	125 -2.04655	175 -2.2733
26 -2.17021	76 -2.10785	126 -2.11585	176 -2.26548
27 -2.14138	77 -2.24404	127 -2.00289	177 -2.08044
28 -2.23387	78 -2.26121	128 -1.99987	178 -2.20038
29 -2.08206	79 -2.0954	129 -2.06011	179 -2.08312
30 -2.1722	80 -1.94822	130 -1.88198	180 -2.23634
31 -2.07496	81 -2.07479	131 -2.04546	181 -2.12257
32 -2.21496	82 -2.02514	132 -1.92965	182 -2.00982
33 -2.08869	83 -2.00264	133 -1.93937	183 -2.04966
34 -2.13923	84 -2.07962	134 -1.97606	184 -2.10712
35 -2.21053	85 -2.11416	135 -2.20329	185 -2.16954
36 -2.06162	86 -2.11055	136 -2.12475	186 -2.24635
37 -2.15408	87 -2.28909	137 -2.15976	187 -2.06277
38 -2.16102	88 -2.02784	138 -2.15484	188 -2.05304

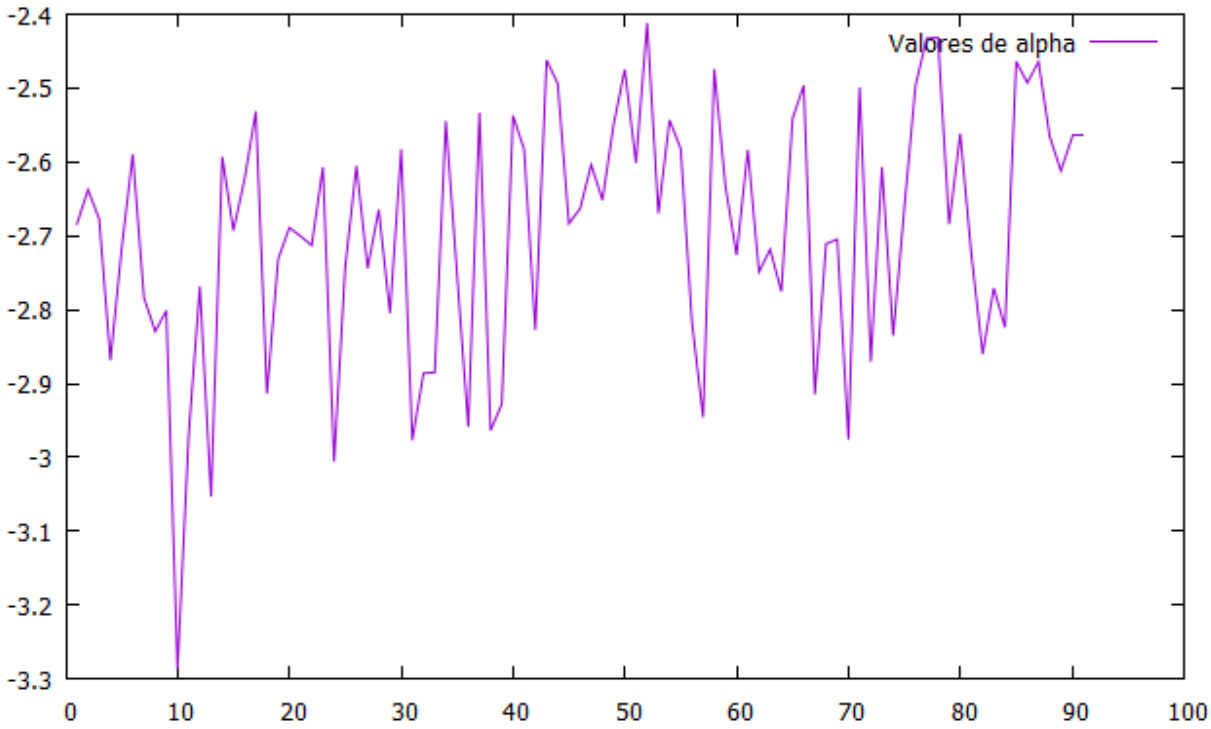
39 -2.06489	89 -2.03219	139 -2.00172	189 -2.11498
40 -2.08531	90 -2.23325	140 -2.05752	190 -2.16315
41 -2.06274	91 -2.08127	141 -2.0484	191 -2.05272
42 -2.08862	92 -2.15989	142 -1.99867	192 -2.08066
43 -2.12147	93 -2.32821	143 -2.14029	193 -2.11555
44 -2.13448	94 -2.22239	144 -1.96084	194 -2.18173
45 -2.03852	95 -2.05073	145 -2.04649	195 -2.18173
46 -2.05127	96 -2.23116	146 -2.10977	196 -2.18173
47 -2.06508	97 -2.03214	147 -2.09861	197 -2.18173
48 -2.10417	98 -2.05519	148 -2.11682	198 -2.18173
49 -2.18432	99 -2.03672	149 -2.0321	199 -2.18173
50 -2.12765	100 -2.02238	150 -2.08993	200 -2.18173



Neste caso, dado que o cenário é mais complexo, não existe um momento em que valor de alfa tenha uma variância considerável, pelo que uma conclusão que se pode retirar é que em cenários mais complexos, o algoritmo não é tão eficaz como num

cenário em que apenas existe um objeto para análise. Mas também pode dizer-se que pelo facto de existirem duas bolas os alfas variam muito, pois o objeto tanto aproxima como está distante no mesmo instante de tempo.

Por último fez-se uma avaliação ao vídeo com a aproximação de um carro (2car.avi), como podemos ver pelo gráfico:



Aqui não se conseguem mais uma vez retirar muitas conclusões com o este algoritmo, pois este ambiente era o mais complexo em relação a todos os anteriores. Desta forma, o que se pode concluir é que como já foi dito para ambientes mais complexos este algoritmo não funciona como o desejado.

De maneira a concluir esta parte da solução, apresenta-se uma *print* da aplicação em execução onde se podem observar os valores de *alpha* para cada *frame*.

Conclusão

Com este trabalho prático ficaram retidos os principais conceitos e técnicas sobre a linguagem de progamação C++ e a biblioteca OpenCV, as quais já tínhamos tido contacto com nas aulas práticas da unidade curricular. O OpenCV demonstrou ser fundamental para a abordagem deste problema, pelo que muitos dos algoritmos usados tiveram que ser redefinidos por nós, não existindo já uma pré-definição dos mesmos.

Como referimos ao longo deste documento, durante o desenvolvimento do projeto enfrentamos vários problemas que nos impediram de avançar mais rapidamente. Depois de algum diálogo com a equipa docente, para tentar perceber melhor o algoritmo e através de algum material fornecido, conseguimos evoluir para uma nova fase do trabalho, em que pudemos avançar e implementar o pretendido.

Podemos dizer que conseguimos implementar uma aplicação totalmente funcional para este problema, embora não tivéssemos incorporado no ambiente de simulação do robô e poder tornar o trabalho mais completo, como nos foi proposto. Esta última etapa foi um problema difícil de abordar, pelo que não tínhamos conhecimento prévio sobre a ferramenta Webots, e tornou-se complicado cumprir este objetivo.

Um aspeto negativo da nossa implementação, foi o excesso de processamento, isto é, apesar de termos usado algumas funções disponibilizadas pelo OpenCV, a maior parte delas, mais importantes e também as mais custosas a nível computacional, foram definidas por nós. Isto tem custos, foi necessário efetuar vários ciclos para iterar dentro das nossas imagens, entre outras coisas e isso evidenciou-se na nossa aplicação final, que demonstra alguma lentidão no processamento da informação. Leva-nos a pensar que talvez o OpenCV incorporado em C++, não seja a melhor solução para realizar este algoritmo de uma forma eficiente. Talvez, outras abordagens como Python ou até o próprio MatLab pudessem ser mais fiáveis no que diz respeito ao desempenho.

No geral, consideramos que o objetivo principal deste trabalho prático foi cumprido com sucesso, e ficamos satisfeitos não só pela aprendizagem durante todo o trabalho mas também pelos resultados finais.

Como trabalho futuro, achamos que poderíamos optimizar a nossa aplicação no que diz respeito ao seu desempenho e também conseguir incorporar a mesma num ambiente de simulação ou mesmo real de um robô.

Bibliografia

1. Ana Carolina Silva, Cristina P Santos, A Time-analysis of the Spatial Power Spectra indicates the proximity and complexity of the surrounding environment, ICINCO 2014, Viena, Aústria
2. http://docs.opencv.org/java/org/opencv/highgui/VideoCapture.html, acedido a 18/12/2014
3. http://docs.opencv.org/doc/tutorials/core/discrete_fourier_transform/discrete_fourier_transform.html, acedido a 18/12/2014
4. http://www.mathworks.com/help/matlab/ref/meshgrid.html, acedido a 04/01/2015
5. http://docs.opencv.org/modules/core/doc/operations_on_arrays.html, acedido a 20/12/2014
6. http://www.mathworks.com/help/matlab/ref/polyfit.html, acedido a 15/01/2015
7. http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html, acedido a 17/01/2015
8. Material de apoio da disciplina disponibilizado pela docente.