Universidade do Minho

Mestrado em Engenharia Informática

Visão por Computador - Computação Gráfica - 2014/2015

Análise Temporal do Spatial Power Spectrum

13 de Fevereiro de 2015

Fábio Gomes pg27752
Hélder Gonçalves pg28505

Índice

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 trabalho que nos foi proposto pela professora Cristina Santos, que tem como objetivo principal implementar em *OpenCV* um trabalho realizado por Ana Carolina Silva do Departamento de Eletrónica Industrial na Universidade do Minho.

O tema principal é centrado na Análise Temporal do Spatial Power Spectra de um vídeo (sequência de imagens) e detectar quando é que o objeto ou objetos estão demasiado perto da câmara de forma a alertar uma possível colisão

Neste documento será descrito todo o trabalho efetuado, bem como as decisões que foram tomadas e as suas dificuldades.

Introdução

Nesta Unidade Curricular um trabalho de grupo faz parte das componentes avaliativas e consiste num projeto sobre um tema proposto, neste caso a Análise Temporal.

Esta Análise Temporal teve que ser feita usando apenas como suporte ao trabalho o documento/*paper* realizado pela Ana Carolina e Cristina Santos.

A percepção visual é muito importante no campo robótico, para efetuar navegações autónomas em ambientes adversos, ou não controlados, e detetar colisões. Infelizmente, a visão é um tarefa excepcionalmente complexa, por isso é tentado simular a Biologia pois é o melhor sistema visual conhecido. A base das pesquisas é feita neste sistema.

Com uma análise de imagens do meio natural podemos perceber que as localizações espaciais vizinhas estão fortemente ligadas em intensidade. Uma aproximação pode ser feita via Transformadas de *Fourier*, o *Power Spectrum* da imagem. As Transformadas são convenientes porque liga as estatísticas da imagem com sistemas lineares para processamento de imagens.

O Power Spectrum de duas dimensões geralmente tem sido reduzida a uma função unidimensional de freqüência espacial, através da realização de uma rotação no plano de Fourier bidimensional. Análises Temporais descobriram que o Power Spectrum de imagens decresce com frequência de $1/f^{\alpha}$, sendo o valor alfa (α) à volta de 2.

As escolhas, os problemas, as soluções, exemplos, tudo será explicado ao longo deste relatório.

Motivação

Com este Trabalho/Projecto é suposto aplicarmos os conhecimentos adquiridos ao longo da disciplina neste semestre e assim conseguir solucionar os problemas de maneira mais rápida e eficaz.

Esperamos ainda que o resultado final esteja dentro do esperado pois o tema é algo novo e foge um pouco à Computação Gráfica em si. Mas só com o decorrer do projecto é que vamos ter uma melhor noção dos desafios que nos irão aparecer e de como os vamos solucionar.

A detecção de movimentos é essencial para a Robótica por exemplo na deslocação de um robô.

Discussão e Implementação do Problema

Antes de começar a fazer código há que aprender e saber ao pormenor o que é que nos é apresentado para delimitarmos os passos a fazer para obtermos o resultado final pretendido.

Os procedimentos a realizar são iguais em cada frame, portanto o esforço computacional será frame a frame. De notar que as imagens têm que ser quadradas.

Tudo começa com a aplicação da Transformada de Fourier (FT) à frame através de:

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

Onde $It\,(x,y)$ refere a intensidade do pixel em (x,y) na imagem na frame número t; fx e fy são as frequência espaciais nas direções x e y; o tamanho da imagem será o N.

Depois calculamos o Power Spectrum (PS) através da seguinte igualdade:

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

Efetuando de seguida uma rotação no plano bidimensional de Fourier. A equação (2) é reduzida a uma dimensão da frequência espacial fr e aproximada pela equação (3):

$$f_r = \sqrt{f_x^2 + f_y^2} \qquad \mathrm{P}_t(f_r) = \mathrm{A}\cdot\frac{1}{f_r^{\alpha_t}} = \mathrm{A}\cdot f_r^{-\alpha_t} \qquad (3)$$

Em que Pt é linear com o declive igual a $-\alpha$, quando desenhado num escala logarítmica e A é uma constante arbitrária que depende da composição da cena.

De forma a analisar a variação do declive ao longo do vídeo podemos saber se o objecto está a aproximar-se ou a ficar mais longe da câmara, é um método indireto e de aproximação:

$$\triangle\alpha_t = \alpha_t - \alpha_{t-1} + \triangle\alpha_{t-1} \qquad (4)$$

$$\begin{cases} \triangle\alpha_t = 0, \text{ if } \triangle\alpha_t < 0 \\ \triangle\alpha_t = \triangle\alpha_t, \text{ if } \triangle\alpha_t \geq 0 \end{cases} \qquad (5)$$

Implementação em C++ e OpenCV

Estudado e percebido o método apresentado chegou a altura de implementar este algoritmo em C++ e OpenCV.

Temos 2 Sistemas Operativos diferentes mas nada disso influencia o comportamento do programa: *OSX Yosemite* com *XCode 6* e *Windows 7* com *Visual Studio 2013* pois o código é portável como é normal.

Antes de tudo é preciso gravar as frames do vídeo, obtemos isso através de:

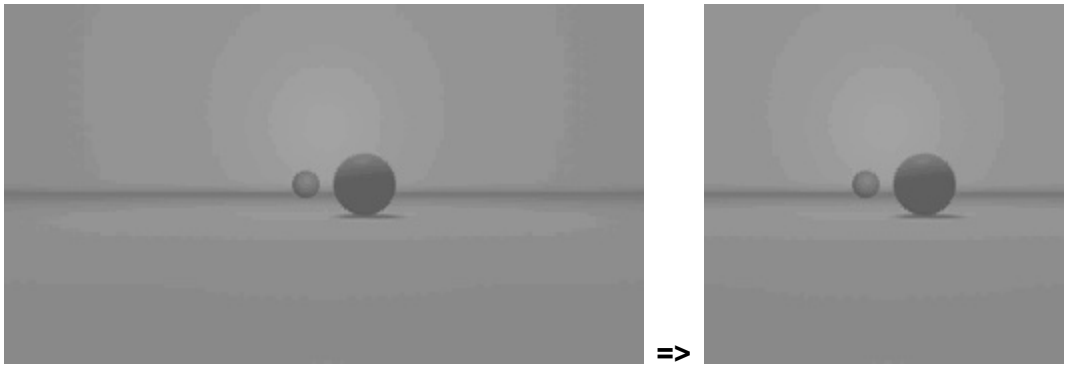
```
VideoCapture cap("videos/2_bolas.avi");
cap >> frame;
cvtColor(frame, edges, CV_BGR2GRAY);
```

agora é só trabalhar com `frame` e aplicar todos os métodos falados anteriormente sobre esta imagem que é alterada a cada iteração/passagem no tempo. Passamos para escala de cinzentos para melhor detetar as variações.

Como as imagens são quadradas e temos que usar DFT nela, podemos fazer um crop à imagem dependendo do tamanho menor das colunas ou linhas (`m`) e mediante esse tamanho usar um valor óptimo (`mdft`) para a o cálculo da Transformada ser mais eficiente.

```
m = frame.rows > frame.cols ? frame.cols : frame.rows;
int mdft = getOptimalDFTSize( m ); arraySize = mdft/2+1;
arraySize = mdft/2+1;
```

Se o tamanho da mdft for maior que o m, um resize será feito à imagem para a nossa dimensão. O `arraySize` corresponde o índice até onde vamos usar o array numa fase seguinte.



Depois de pronta a imagem vamos utilizar a DFT (Transformada de Fourier), mas para isso é preciso criar uma matriz bidimensional (`complexI`, explicada anteriormente) e portanto é criada outra só de zeros para juntar. Faz-se isto com o `merge` e depois `dft`:

```
planes[0] = Mat_<float>(padded);
planes[1] = Mat::zeros(padded.size(), CV_32F);
merge(planes, 2, complexI);
dft(complexI, complexI);
```

Depois calculada a dft e gravado o resultado multi-canal, é preciso separar (`split`) os planos *Real* (`planes[0]`) e o *Imaginário* (`planes[1]`). A *Magnitude* será calculada com o `magnitude` e ficamos na Matriz `magI` o resultado.

```
split(complexI, planes);
magnitude(planes[0], planes[1], planes[0]);
magI = planes[0];
```

Convertemos a escala para Logaritmo da seguinte forma:

```
magI += Scalar::all(1);
log(magI, magI);
```

e depois cortamos o Spectrum se tivermos um número ímpar de linhas ou colunas e centramos a imagem de Fourier com o centro da frame reajustando os quadrantes.

```
magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));
fftShift(magI);
```

Já temos a imagem tratada falta agora gerar o histograma das frequências. Para isso usamos a seguinte função bastante optimizada e eficiente.

```
void findRho_impf(int *num, float *f){
   int i,j, r;
   for (i=0; i<rho.cols; i++) {
       for (j=0; j<rho.rows; j++) {
          r=(int)rho.at<float>(j, i);
          if(r>0 && r<arraySize){
             f[r] += powf(magI.at<float>(j, i), 2);
             num[r]++;}
        }}
   for (r = 1; r<arraySize; r++)
      if(num[r]!=0)    f[r] /= num[r];
}
```

Daqui obtemos com o array f a lista das médias dos quadrados da magnitude para cada rho.

Tendo a lista `f` calculada no passo anterior, adicionamos a uma lista de pontos cuja posição no eixo dos x é o logaritmo do `i` que vai de 1 até `arraySize` e com logaritmo do `f`.

Tendo esta lista de pontos faz-se o `fitLine` para aproximar a distribuição a uma recta. Baseado nessa recta calculamos o declive da mesma usando o vector colinear que a define. Esse declive será o `alpha` que estamos à procura. Esse `alpha` irá para o gráfico temporal para depois se efectuar a Análise.

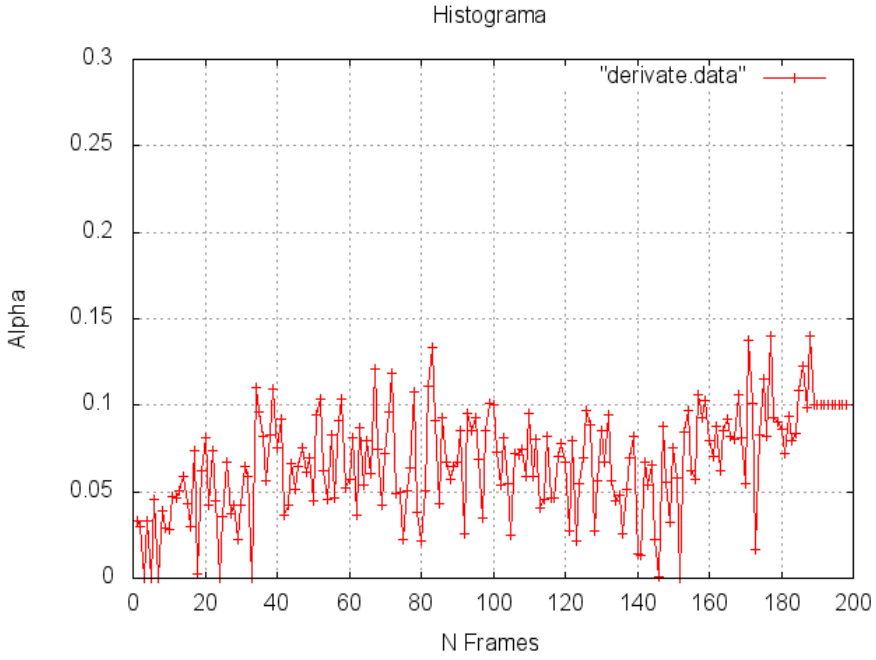
```
for (i=1; i < arraySize;i++)
    points.push_back(Point2f((float)log(i), (float)log(f[i])));

fitLine(points, line, CV_DIST_L2, 0, 0.01, 0.01);
alpha = 0.0f;
alpha = line[1] / line[0];

if(alpha<0)
     add_alpha(alpha);
else
     add_alpha(0);
```

Acabado o ciclo podemos limpar os dados em memória inicializando-os para a próxima iteração/frame. Incrementamos o número de frames.

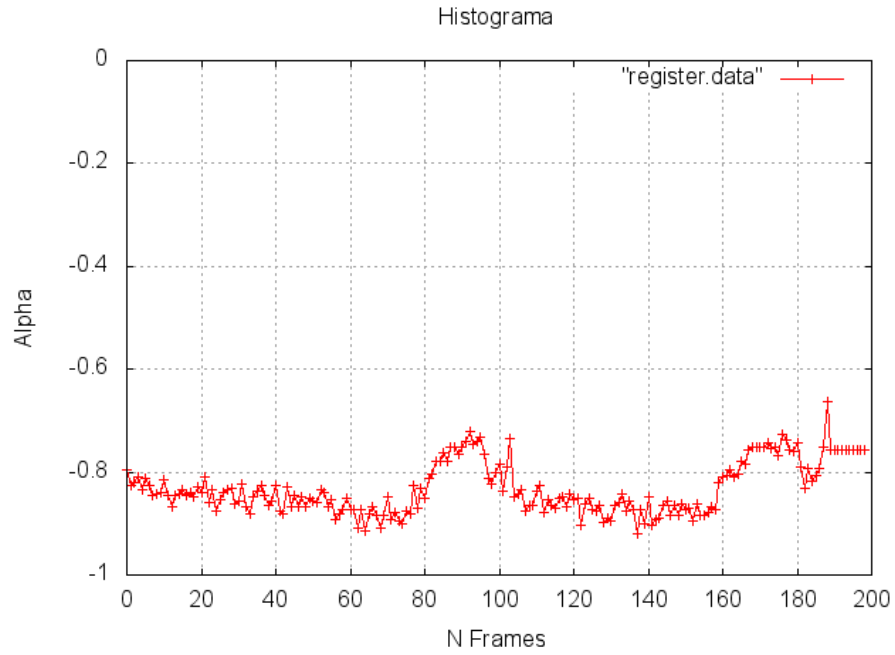
```
memset(f, 0, arraySize * sizeof(float));
memset(num, 0, arraySize * sizeof(int));
points.clear();

cout<< framess++ <<endl;
```

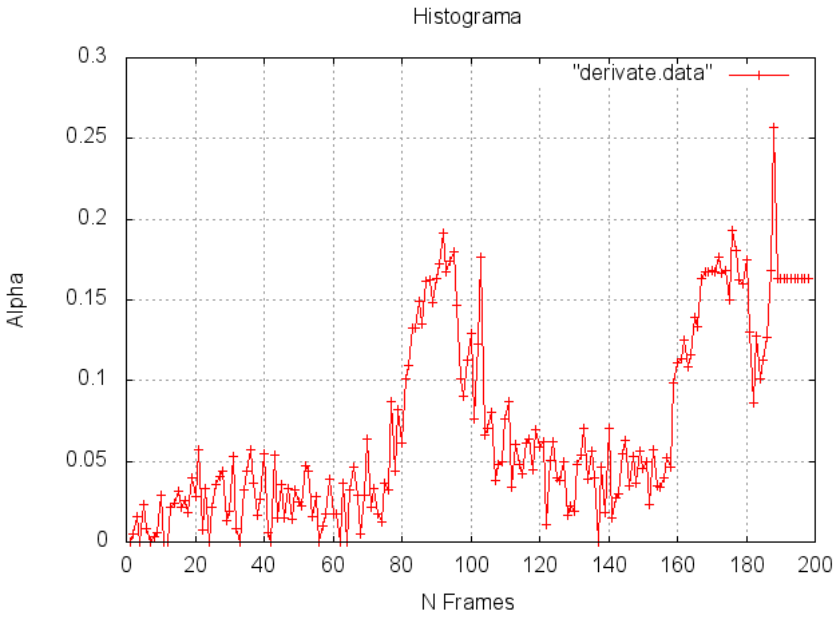
Comparação de Resultados

Pegando por exemplo no vídeo "*2_bola.avi*" tivemos o seguinte gráfico de alfas.



É perceptível que os dados que podemos obter sobre o objeto ao longo do tempo são praticamente nulos. O gráfico serve de nada. Depois de falarmos com a Ana Carolina ela sugeriu-nos usar um filtro Gaussiano. Aplicado o Filtro (11,11) ficamos com um resultado melhor.



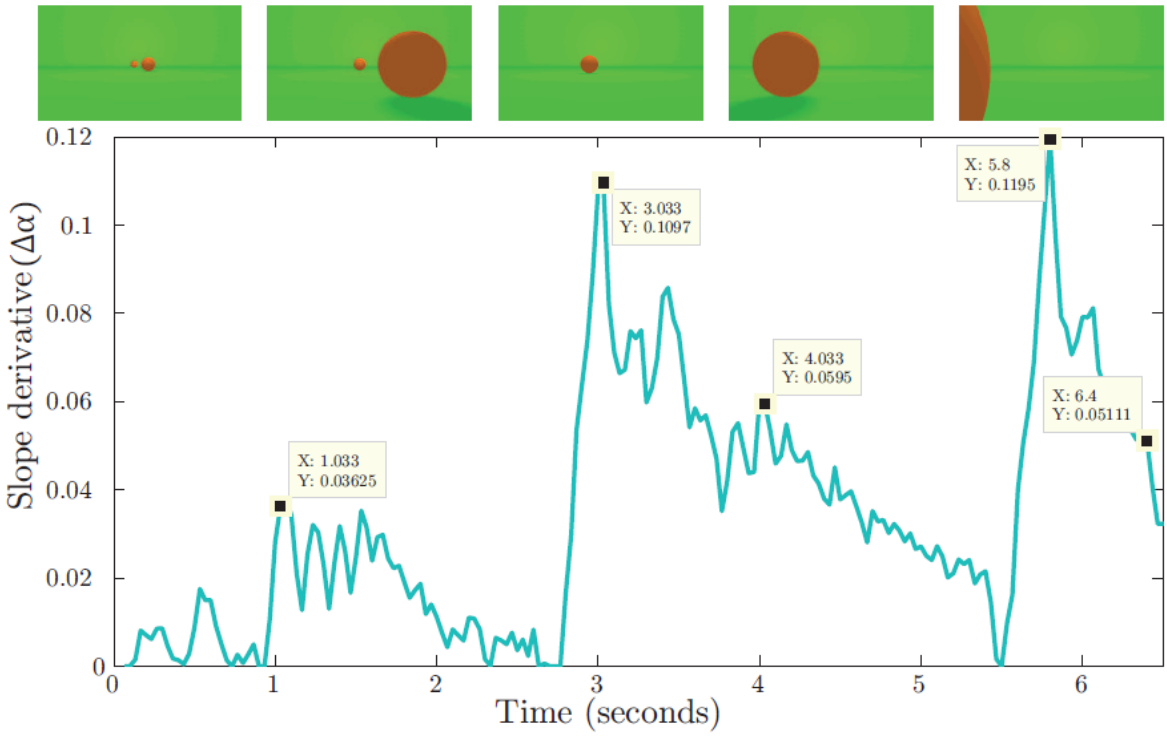


Portanto passaremos a usar o Filtro Gaussiano para todas as frames antes da DFT.

```
GaussianBlur(padded, padded, Size(11,11), 1);
```

O que este filtro vai fazer é atenuar as altas frequências da imagem que, normalmente, não são tão importantes como as baixas frequências. Os obstáculos normalmente possuem baixas frequências e é neles que estamos mais interessados.

Comparando com o resultado dos declives (última figura) com o do paper para o mesmo vídeo dá para perceber que obtemos resultados muito semelhantes.



A nível de tempos conseguimos resultados muito satisfatórios, para vídeos semelhantes aos divulgados conseguimos em tempo real processar teoricamente (usando os valores da tabela seguinte *1s/0.011 ~= 90*) processar **90 frames por segundo** (para 180 linhas). Ou seja, **3x** mais rápido em comparação com os 30 fps.

Ou seja, a nossa implementação pode ser usada em tempo real.

Tempos

Vídeo	#Frames	#Duração (segundos)	#Tempo de Processamento	#segundos / frame
car2.avi	200	3	1.7s	0.0085
approaching_lv_40ms_translate_approach.avi	228	7	4.1s	0.0179
1_bola.avi	200	6	2.2s	0.011
2_bolas.avi	200	6	2.2s	0.011

Tudo isto tem que ser devidamente aproveitado e só faz sentido se conseguirmos, por exemplo, avisar o robô que um objecto está a aproximar-se ou algo está a acontecer que ele precise de saber. Foi criada uma componente que com os valores das derivadas obter tais

dados, chama-se *Algoritmo de Detecção de Movimento*. Ficamos assim com uma parte de detecção de acontecimentos como Movimentos, Colisões e Sensação de Perto. Estes tempos já incluem esse método pois é importante contabilizá-lo.

Análise de Resultados

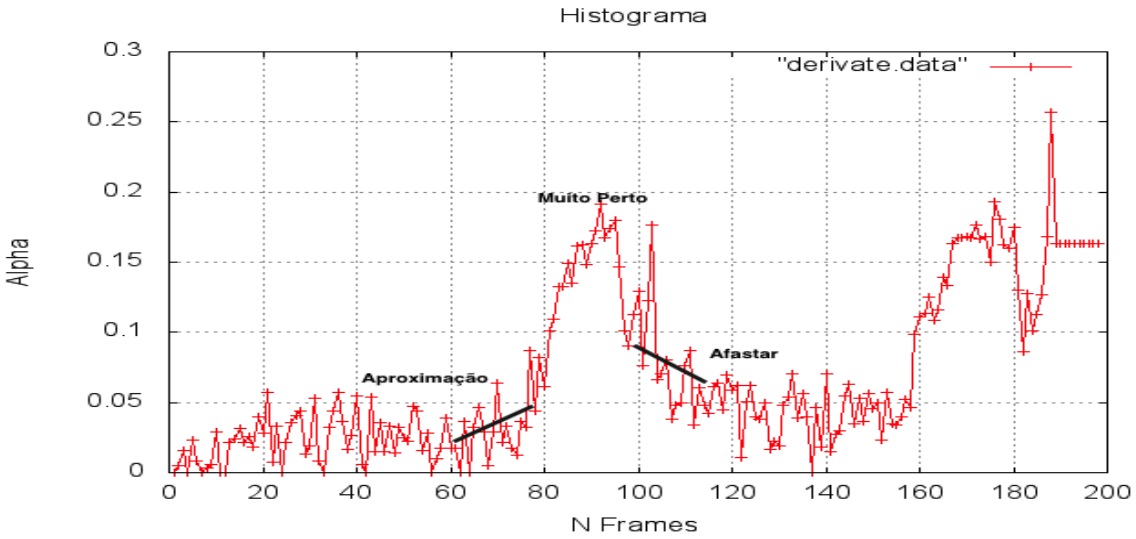
A análise de resultados é um dos principais objectivos deste projecto. A identificação/detecção de movimentos é bastante importante, para possibilitar o envio de sinais para outro sistema e assim este tomar as melhores opções dependendo do resultado obtido. Os momentos que mais interessam detecção são se o um objecto se está em identificar a trajetória de um objecto, que pode ser uma simples translação, aproximação ou afastamento. A detecção se o objecto está perto da câmera ou se existe uma colisão também são detecções de interessantes a tomar em conta, chamamos de *Algoritmo de Detecção de Movimento*.

Algoritmo de Detecção de Movimento

A análise tem em consideração os gráficos das derivadas do *alpha*. O programa manda um sinal para o utilizador quando detecta alguns movimentos na imagem. Quando não existe nenhuma variação no alpha em relação à frame anterior o programa não vai mandar nenhum sinal para o utilizador, pois significa que a imagem se manteve constante e não há nada a reportar.

Pela análise dos gráficos obtidos, apercebemo-nos que sempre que existia movimento na imagem os alfas sofriam oscilações. São estas oscilações o foco importante mais importante para a detecções de movimentos neste trabalho. Quando as oscilação são de pequena grandeza, significa que algo se está a passar no ambiente. Nestes casos o nosso programa é capaz de identificar algumas trajectórias básicas. A aproximação e afastamento de objectos são as trajectórias identificadas pelo programa. No caso de detectar algum movimento mas nenhuma destas trajectórias ser reportada ele simplesmente avisa que algo se está a movimentar, mas não conseguiu identificar. Agora, como é que estamos a detectar as trajectórias!? Apercebeu-se que sempre que um objecto se aproximava da câmara que o valor da derivada aumentava e que diminuia caso se afastasse. Posto isto, recorre-se ao histórico das últimas frames, neste momento as últimas 10, e traçamos uma linha que encaixe nos pontos obtidos. Assim, de uma forma genérica, se o declive desta recta for positivo estamos perante uma aproximação, caso contrário estamos perante um afastamento. Claro que se a recta for negativo não é obrigatoriamente um afastamento, por temos uma pequena tolerância, e se o declive não é acentuado o suficiente o programa só reporta um houve alguma alteração, mas não prevê a trajectória.

No exemplo a seguir temos o video ‘2_bolas.avi’, e vemos em detalhe o que está a acontecer na análise do gráfico. Temos também o output dos sinais dados pela aplicação para este video, no momento em que passa a primeira bola.

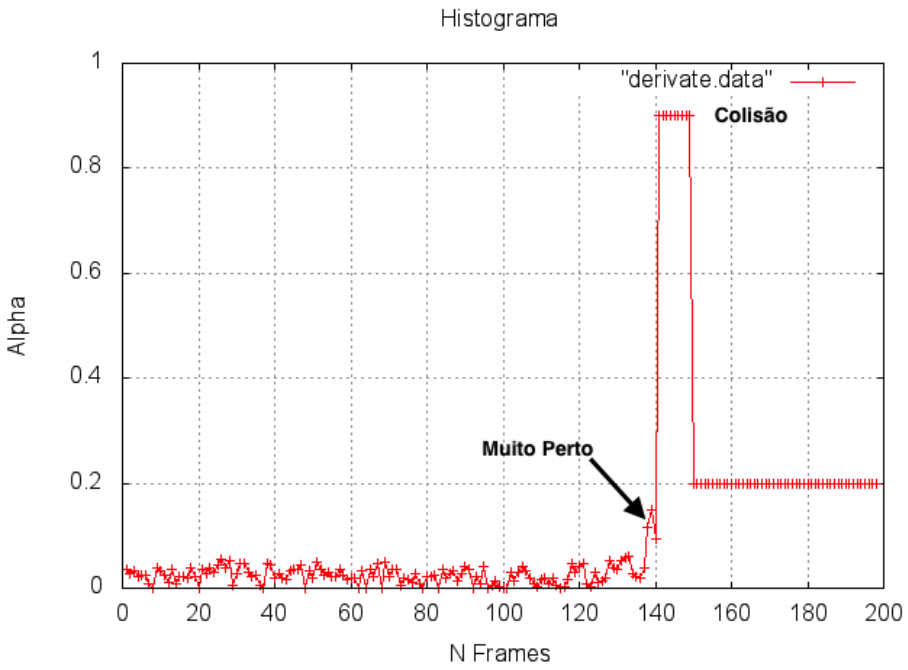



```
66 -->Algum Movimento!!
71 -->Algum Movimento!! - Aproximar
72 -->Algum Movimento!! - Aproximar
78 -->Algum Movimento!! - Aproximar
79 -->Algum Movimento!! - Aproximar
80 -->Algum Movimento!! - Aproximar
82 -->Muito PERTO!!
84 -->Muito PERTO!!
88 -->Muito PERTO!!
94 -->Muito PERTO!!
97 -->Muito PERTO!!
98 -->Muito PERTO!!
100 -->Muito PERTO!!
102 -->Algum Movimento!!
103 -->Muito PERTO!!
104 -->Muito PERTO!!
105 -->Algum Movimento!! - Afastar
108 -->Algum Movimento!! - Afastar
113 -->Algum Movimento!!
123 -->Algum Movimento!!
```

Como se pode ver no output de sinais dado, existe também outro sinal, 'Muito PERTO!!'. Este sinal está sobreposto aos anteriores. Ele é desperto, assim como os anteriores, quando existe uma variação significativa do valor do alpha, mas além dessa condição também é necessário que estejam a ser reportados valores elevados do alpha.

Para este sinal é sempre utilizado como base da análise, os gráfico obtdo a partir dos alpha's. Mas na nossa opinião, este não seria o melhor método. A melhor aplicação seria em primeiro lugar, a partir dos métodos de morfologia, aumentar o contraste da imagem, pois normalmente existe muito pouco contraste nestas imagens. Depois em segundo lugar aplicavamos um *threshold* e a seguir identificávamos os diferentes objectos e víamos a área da imagem que eles ocupavam. Quanto maior fosse, mais próximo estaria o objecto.

Além destes quatro eventos descritos, a nossa aplicação ainda é capaz de detectar colisões. Esta tem como base a análise feita para o sinal 'Muito PERTO', mas a grande diferença é que para ser reportada uma colisão é necessário que as variações do alpha sejam muito mais intensas. Podemos ver esse exemplo no video da '1_bola.avi'. Em que nos é repostada uma colisão.

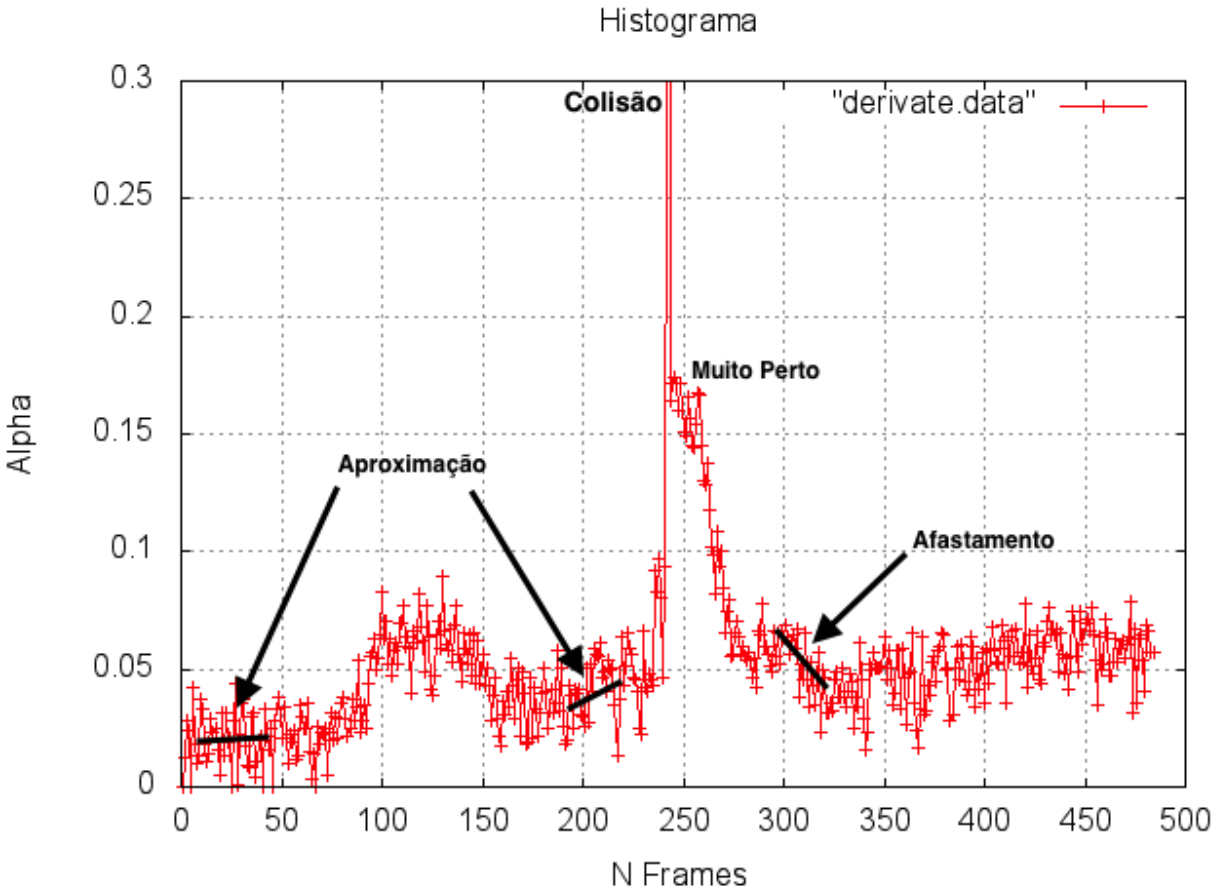



```
52 -->Algum Movimento!!
64 -->Algum Movimento!! - Aproximar
65 -->Algum Movimento!!
66 -->Algum Movimento!!
69 -->Algum Movimento!! - Aproximar
70 -->Algum Movimento!! - Aproximar
74 -->Algum Movimento!!
85 -->Algum Movimento!!
93 -->Algum Movimento!!
96 -->Algum Movimento!!
97 -->Algum Movimento!! - Aproximar
119 -->Algum Movimento!! - Aproximar
123 -->Algum Movimento!!
129 -->Algum Movimento!!
135 -->Algum Movimento!!
139 -->Muito PERTO!!
140 -->Muito PERTO!!
141 -->Muito PERTO!!
142 -->COLISÃO!!
151 -->COLISÃO!!
```

Como podemos ver ele reporta duas colisões. Isto está não está 100% correto, mas acontece porque existem duas grandes variações do *alpha*, uma positiva (quando colide) e outra negativa (quando a bola sai da imagem). E podemos ainda ver que antes de serem disparadas as colisões ele avisou que estava muito perto de um objecto. Também é de notar que o programa não consegue ter muita exatidão em relação à aproximação da bola, pois como podemos ver tem muitos movimentos que ele não consegue identificar exatamente o que está a acontecer.

Até agora vimos vídeos em que o ambiente estava relativamente favorável, mas não será este tipo de ambiente final que nos interessa. Para isso, fizemos um pequeno video caseiro, a partir da *webcam* do nosso próprio computador para ver como é que ele se ia comportar.

Este video é um pouco mais mais pesado que os anteriores devido à sua dimensão, 720 linhas, mas mesmo assim ainda conseguimos bons desempenhos no algoritmo. Neste ambiente as condições são muito mais complicadas que as dos teste anteriores, fazendo com que algumas situações o algoritmo de análise não se conseguiu obter os melhores resultados. A maior dificuldade para ele, é detectar as trajectórias correctas, pois este fica-se só por identificar algum movimento.

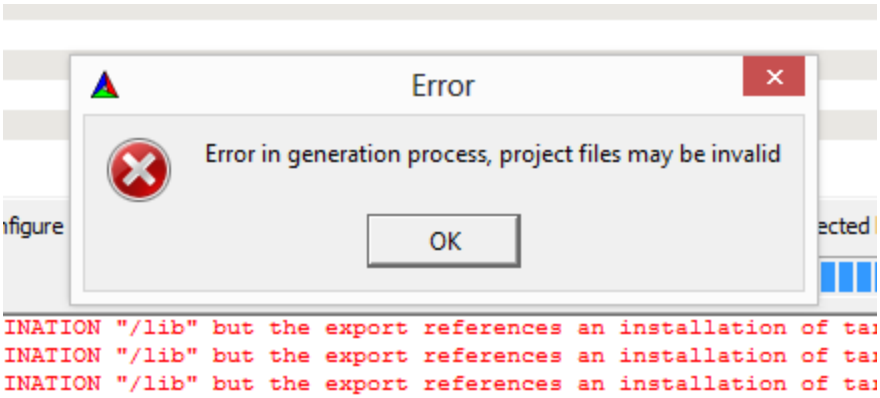



```
7 -->Algum Movimento!!
29 -->Algum Movimento!!
43 -->Algum Movimento!! - Aproximar
90 -->Algum Movimento!!
231 -->Algum Movimento!! - Aproximar
237 -->Muito PERTO!!
241 -->Algum Movimento!!
242 -->Muito PERTO!!
243 -->COLISÃO!!
245 -->COLISÃO!!
251 -->Muito PERTO!!
260 -->Muito PERTO!!
268 -->Muito PERTO!!
319 -->Algum Movimento!! - Afastar
370 -->Algum Movimento!!
371 -->Algum Movimento!! - Aproximar
422 -->Algum Movimento!! - Aproximar
475 -->Algum Movimento!!
476 -->Algum Movimento!!
```

Podemos ver na imagem ao lado os sinais dados pela aplicação em relação ao demo fornecido. Podemos ver que ele se comportou relativamente bem neste ambiente. Teve um momento em que não houve sinais, a partir da frame 90 até à 231. Isto deve-se à dificuldade que o algoritmo tem em reagir a pequenas variações na imagem. Podemos ver que nesse intervalo, não houve nenhum sinal apesar de nós, humanos, conseguirmos identificar naquela zona o efeito de uma parábola, que se deve a uma aproximação lenta de um objecto à câmera, e e seu afatamento de seguida. Na segunda aproximação, mais rápida que a anterior verifica-se que ele identificou a sua aproximação, e um momento antes da colisão, alertou que estava muito perto. Depois, continuou bastante perto da câmera e vemos que é identificado o afastamento do objecto.

Simulação em Webots

Para uma melhor análise do nosso algoritmos foi opcionalmente apresentado o *Webots* para simular um ambiente com um robô. Para conseguirmos usá-lo era necessário compilar as bibliotecas do *OpenCV* com o *CMake* usando o compilador do *Code::Blocks*. Só que não conseguimos fazê-lo pois dava o seguinte erro a compilar o OpenCV e não houve forma de o conseguirmos resolver:



Este problema não é grave pois o *Algoritmo de Detecção de Movimento* está adaptado para funcionar nesse ambiente, tendo apenas de alterar o modo como avisa os movimentos. Para este caso em que só temos que detetar movimentos, o problema de não termos conseguido não é tão grave pois o mais importante está feito.

Solução e Validação

Com o algoritmo implementado testamos várias vezes e validando sempre com o código Matlab fornecido para consulta pela Ana Carolina. As validações foram feitas passo a passo à mão correndo os comandos equivalentes no Matlab com os mesmos dados e obtemos sempre os mesmos resultados. Até que chegou à parte final em que é usado a *polyfit* no Matlab e a *fitLine* no openCV deram valores muito próximos pois a sua implementação é evidentemente diferente.

Portanto não é um problema aparente da nossa implementação. Temos tudo confirmado e a funcionar.

Por fim o *Algoritmo de Detecção de Movimento* faz a sua análise, muito relativa, não estando perto sequer de uma implementação *fail-proof*. Trata-se apenas de uma simples implementação de análise de variância de alfas nos frames anteriores e portanto não deverá ser alvo de testes de validação intensivos. Funciona para muitos casos mas também alerta para eventos que são incorrectos. No geral estamos bastante satisfeitos com o resultado por ele alcançado pois já nos obrigou a fazer uma análise subjectiva das variâncias dos alfas ao longo dos frames.

Conclusão

Finalizado o Trabalho podemos concluir que estavamos à espera de algo mais relacionado com a matéria da Unidade Curricular. Apenas nos limitamos a fazer aplicação de um algoritmo que nos foi apresentado como funcional. As aulas ao longo do semestre pouco serviram para nos ajudar neste trabalho.

A única fonte de informação era o paper e tínhamos que seguir o método fórmula a fórmula. O problema é que essas fórmulas não eram devidamente explícitas e ficamos várias vezes bloqueados pois não entendíamos o que elas faziam ou precisavam. Falamos mais que uma vez com a Ana Carolina para nos auxiliar nas abordagens pois não havia ninguém mais dentro do assunto.

Depois de nos juntarmos com os outros grupos deste mesmo tema e de mais umas trocas de e-mail com a Ana conseguimos avançando tendo como ajuda o código Matlab por ela facultado. Assim ficou mais fácil avançar e entender o que era pedido.

A nível de implementação em si, não tivemos muitas dificuldades pois C++ e OpenCV são bastante fáceis de utilizar. O problema como referido foi no entendimento do projecto em si e de tudo o que ele acarretava. O que deu mais gosto fazer até foi o último passo, a análise para detecção de movimentos em que criamos o *Algoritmo de Detecção de Movimento*.

No final os gráficos eram semelhantes aos do paper mas não chegam à precisão centesimal que o mesmo apresenta. Ficamos duvidosos daqueles valores pois mostravam linhas quase perfeitas como uma função exponencial e nós com o algoritmo acabado não chegavamos a esses resultados. Os valores de alfa para os vídeos divulgados variam entre -0.5 e -0.8 e nunca perto dos -2 anunciados. Os dos declives ainda pior são, tanto dão muito altos como muito baixos, mas isso supomos que se deva à discrepância do ambiente dos vídeos.

Resumindo, estavamos à espera de um trabalho mais no âmbito da Computação Gráfica “aplicada” e não tanto a tentar implementar um algoritmo que até já estava feito e funcional. Gostaríamos de ter problemas em tentar detectar algo com a imagem, em vez disso era com problemas a nível dos métodos do paper.

As dúvidas foram resolvidas e bem explicadas pela Ana Carolina nas sessões que tivemos que marcar e graças a isso conseguimos avançar bem com o projecto.

O balanço final não foi muito positivo pois esperávamos mais de um projecto final de UC num Mestrado. Todo o trabalho que se fez ao longo do semestre devia culminar com o Projecto e tentar aplicar todos os conhecimentos que adquirimos. Em relação ao trabalho foi talvez dos trabalhos apresentados o que menos gostaríamos, nem hipótese de escolha tivemos pois foram sorteados.

O Trabalho foi concluído mas não com o grau de aprendizagem esperado destes projectos.

Bibliografia

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