Universidade Técnica de Lisboa
Instituto Superior Técnico

Licenciatura em Engenharia Física Tecnológica

"Estatística de Dados"

Processamento Automático da Contagem de Dados por Análise de Imagem

Projecto de Fim de Curso de: Pedro Jorge de Paula Carvalho; Nº 46752
Professor Orientador: Horácio Fernandes

Índice

1 – Introdução

1.1 – Objectivos

Este projecto consiste no desenvolvimento de um sistema capaz de processar uma imagem que contenha vários dados de jogar, de modo a contar as pintas presentes na imagem. É possível baralhar os dados automaticamente, de modo a obter imagens diferentes para contagem.
Com suficientes contagens é possível construir uma estatística com um valor médio e um desvio padrão - distribuição gaussiana - que poderá ser comparada com a obtida em contagens de um contador Geiger.
Este projecto faz parte do programa E-Lab e, como tal, foi desenvolvido de acordo com esta plataforma, no que diz respeito à colocação na web.

1.2 - Projecto e-escola

1.2.1 - Apresentação

O portal e-escola é um portal de educação em ciências básicas na Internet, que pretende fazer a ponte entre o ensino secundário e as exigências da Universidade nas disciplinas de Matemática, Física e Química. dirigido quer a professores como alunos. É uma iniciativa da UTL através do Instituto Superior Técnico.

O portal está disponível no endereço electrónico www.e-escola.utl.pt e está acessível a todos os interessados, embora o público-alvo seja o grupo constituído por alunos e professores do ensino secundário.
Trata-se de um portal com formas de visualização diferentes para alunos, professores e escolas, que pretende funcionar como um complemento do ensino que se faz no secundário nas áreas da Física, Química e Matemática. Estas são as áreas escolhidas, por serem as áreas para as quais existe uma maior necessidade de atrair alunos em Portugal.
Este portal conta com o apoio da União Europeia, via POSI (Programa Operacional para a Sociedade da Informação) e PRODEP III (Programa de Desenvolvimento Educativo para Portugal).
O projecto pode ser útil tanto para os alunos com dificuldades em acompanhar a matéria nas aulas como um espaço para estimular os mais brilhantes, já que a informação está disponível em vários níveis de complexidade.
Além dos conteúdos (em formato texto, vídeo, animação, gráficos), alunos e professores podem ainda realizar, online, experiências preparadas pelos investigadores do IST, muitas vezes demasiado caras ou perigosas para fazer numa sala de aula. As experiências estão totalmente robotizadas de modo a que o visitante possa realizar, em tempo real, aquilo de que só tem um conhecimento teórico.
Estão previstos mecanismos interactivos como a resposta a perguntas por parte dos docentes do IST ou a possibilidade de o aluno realizar on line testes sobre determinada matéria, que são automaticamente corrigidos. Se a amostra de visitantes for suficientemente alargada, estes testes podem ser indicadores preciosos para o Ministério da Educação, permitindo detectar eventuais falhas

na aprendizagem de determinadas matérias. Para os professores está também prevista a disponibilização de ferramentas auxiliares de ensino em formato multimédia. Notícias, curiosidades e factos sobre personalidades relevantes do mundo científico são outras das áreas que completam este portal, um espaço que se espera vir a alcançar cem mil acessos ("page views") por mês.

1.2.2 - eLab

O “eLab” é um espaço, dentro do e-escola, onde se podem realizar experiências reais através da Internet. As experiências estão montadas e instaladas fisicamente num laboratório do Instituto Superior Técnico. Cada experiência é controlada pelo administrador da experiência que não é mais do que o primeiro membro da lista de espera dos interessados em efectuá-la. No entanto existe uma lista dos “clientes” dos dados que são todas as pessoas visitantes da página. Isto permite por exemplo que um professor realize a experiência e os seus alunos recebam em simultâneo a imagem e os dados, apesar de poderem estar em locais fisicamente distantes.

Os dados são captados por meio de sensores conectados directa ou indirectamente a um computador central de onde são difundidos através da Internet bem como a imagem dos acontecimentos.

Cada sala de controlo corresponde a uma determinada experiência e dispõe de um “chat” on-line onde todos poderão tecer comentários e trocar informação sobre a experiência e sobre a análise dos dados. Cada sala dispõe ainda dum espaço próprio, uma lição, onde se sugere um protocolo experimental, sugestões de variantes à experiência e as explicações e análises que podem ser dadas por todos os utentes do portal.

2 – Teoria

2.1 - Vibração da Plataforma para Baralhar os Dados

Vários artigos já foram escritos acerca duma plataforma vibradora com uma partícula sobre ela e o comportamento desta. Entre esses destaca-se "A. Mehta and J. M. Luck, *Novel Temporal Behavior of a Nonlinear Dynamical System: The Completely Inelastic Bouncing Ball*, Phys. Rev. Let. **65**, Nr.4, 393 (1990)" que usa um modelo simplificado deste sistema, mas, mesmo assim, é um modelo não linear apenas com solução numérica e um pouco complicado.
Para este trabalho, apenas se pretende obter a ordem de grandeza das frequências de vibração da plataforma que sejam capazes de baralhar os dados. Assim, desenvolveu-se um modelo ainda mais simplificado, baseado nas seguintes suposições (algumas das quais também se encontram no modelo do artigo supracitado):

- O dado que parte do repouso acompanha o movimento ascendente da plataforma;
- O dado apenas se separa da plataforma quando esta começa a descer, ou seja, o dado fica em queda livre, com velocidade inicial vertical nula;
- O choque do dado com a plataforma é inelástico, ou seja, volta a acompanhar a plataforma até à seguinte posição máxima desta.

Com estas suposições, podem-se descrever as trajectórias do dado e da plataforma com as seguintes fórmulas (com t a contar a partir do momento que o dado é largado):

- Trajectória do dado: $y(t) = \frac{A}{2} - \frac{gt^2}{2}$
- Trajectória da plataforma: $y'(t) = A\cos(2\pi f t)$

Em que A é a amplitude da oscilação da plataforma, g é a aceleração da gravidade (9,81 ms^{-2}) e f é a frequência da oscilação.

Determinando o tempo em que as duas trajectórias se encontram, chega-se à seguinte equação implícita:

$$t = \sqrt{\frac{A - 2A\cos(2\pi f t)}{g}} \qquad \text{(eq. 1)}$$

Resolvendo, recorrendo a métodos numéricos, para frequências inteiras, entre 1 e 500 Hz, e amplitude da vibração de 3 mm, pode-se construir a seguinte tabela:

f (Hz)	t (miliseg.)	$\cos(2\pi f t)$	$-\sin(2\pi f t)$
14	27.91	-0.7732	-0.6341
15	29.70	-0.9418	-0.3362
16	30.24	-0.9948	-0.1017
17	30.25	-0.9960	0.0893
18	29.98	-0.9692	0.2462
19	29.54	-0.9268	0.3756
20	29.01	-0.8758	0.4827
21	28.42	-0.8206	0.5715
49	30.21	-0.9924	0.1227
50	30.26	-0.9968	0.0805
51	30.06	-0.9781	0.2083
83	30.26	-0.9973	0.0736
116	30.27	-0.9977	0.0680
149	30.27	-0.9980	0.0632
182	30.27	-0.9983	0.0591
215	30.27	-0.9985	0.0554
248	30.28	-0.9986	0.0521
281	30.28	-0.9988	0.0491

Tabela 1: Frequências para as quais o método numérico de resolução da eq. 1 convergiu, com os respectivos valores de t e cos e sin do resultado.

O valor inicial de t utilizado no método foi de $t = {}^{1}\!/_{f}$, de modo a que o coseno, dentro da raiz, dê zero e, portanto, a raiz apresente um valor real (pelo menos na primeira iterada). Para todas as frequências que não constam da tabela, o método ou divergiu ou convergiu para um número complexo.
De notar que, regra geral, as trajectórias cruzam-se perto do mínimo da trajectória da plataforma ($\cos(2\pi f t) \approx -1$), já quando esta vai a subir ($-\sin(2\pi f t) > 0$).

Na prática, constata-se que:

- Para frequências inferiores a 20 Hz, a plataforma não vibra, dá uns estalidos. Isto acontece porque, ou o altifalante que produz o som não tem resposta para estas frequências, ou o amplificador utilizado não tem resposta, ou a placa de som do computador não é capaz de as produzir como sinusoides, ou uma conjugação destes três factores.
- Para frequências entre os 20 e os 150 Hz, a plataforma vibra e os dados são baralhados. Nota-se que, para 50 Hz, o sistema entra em ressonância, sendo possível que, depois de parar a vibração, fique um dado sobre o outro.
- Para frequências superiores a 150 Hz, a plataforma vibra, mas, devido à sua inércia, essa vibração tem uma amplitude bastante reduzida, não sendo frequências capazes de baralhar os dados.

2.2 - Algoritmos utilizados

Antes de qualquer análise de imagem, é necessário transformar a imagem adquirida, para que ela se torne mais fácil de analisar. Este tratamento consiste, essencialmente, na remoção de pormenores da imagem irrelevantes para a posterior análise.
Neste caso, o tratamento aplicado consiste na detecção dos contornos mais significativos da imagem, ou seja, pretende-se obter uma imagem que apenas apresenta os contornos dos dados e das suas pintas.

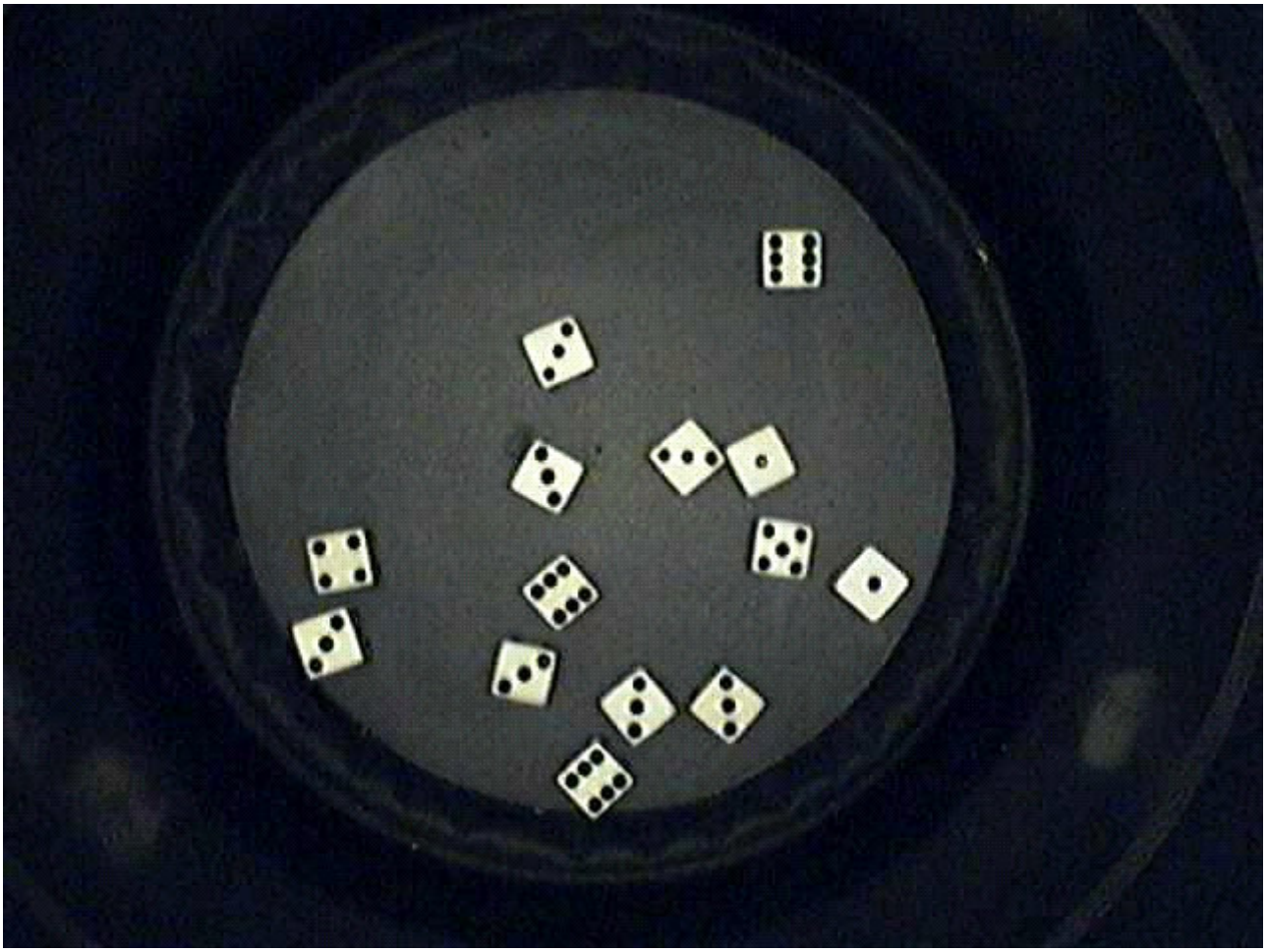


Imagem 1: Exemplo de uma imagem, tal como é adquirida pela WebCam.

O primeiro passo para a obtenção desta imagem é a conversão da imagem original para preto e branco. Assim, ficamos perante uma imagem binária (um pixel ou está a preto ou está a branco) cujo tratamento é extremamente mais simples e rápido. Esta conversão é feita do seguinte modo: calcula-se a média dos valores (r,g,b) de cada pixel e, se essa média estiver acima dum certo valor (*ThresholdBW*), na nova imagem, o pixel fica branco; caso contrário, o pixel fica preto. O *ThresholdBW* é determinado, tendo em conta a iluminação presente e de tal modo a que, na imagem final, apenas haja o fundo, os dados e as suas pintas. Como se estão a usar dados brancos com pintas pretas e o fundo também é preto, este *ThresholdBW* pode ser bastante elevado, mas

pretende-se manter os círculos o mais circulares possível, logo este valor não deve ser assim tão elevado, chegando-se, por tentativas, ao valor de 105.
Este passo é bastante rápido na maioria dos computadores, dado que se está a "varrer" um vector com 640x480 = 307200 entradas. Em cada uma delas fazem-se duas somas, uma divisão, uma comparação de inteiros e actualiza-se um valor num outro vector com a mesma dimensão. Para os processadores actuais isto implica 4 instruções (ou menos) por cada pixel. Assim, para a imagem toda, serão precisos 4x307200 = 1228800 instruções. Um processador cujo relógio seja de 250MHz conseguirá converter 200 imagens por segundo, no mínimo.

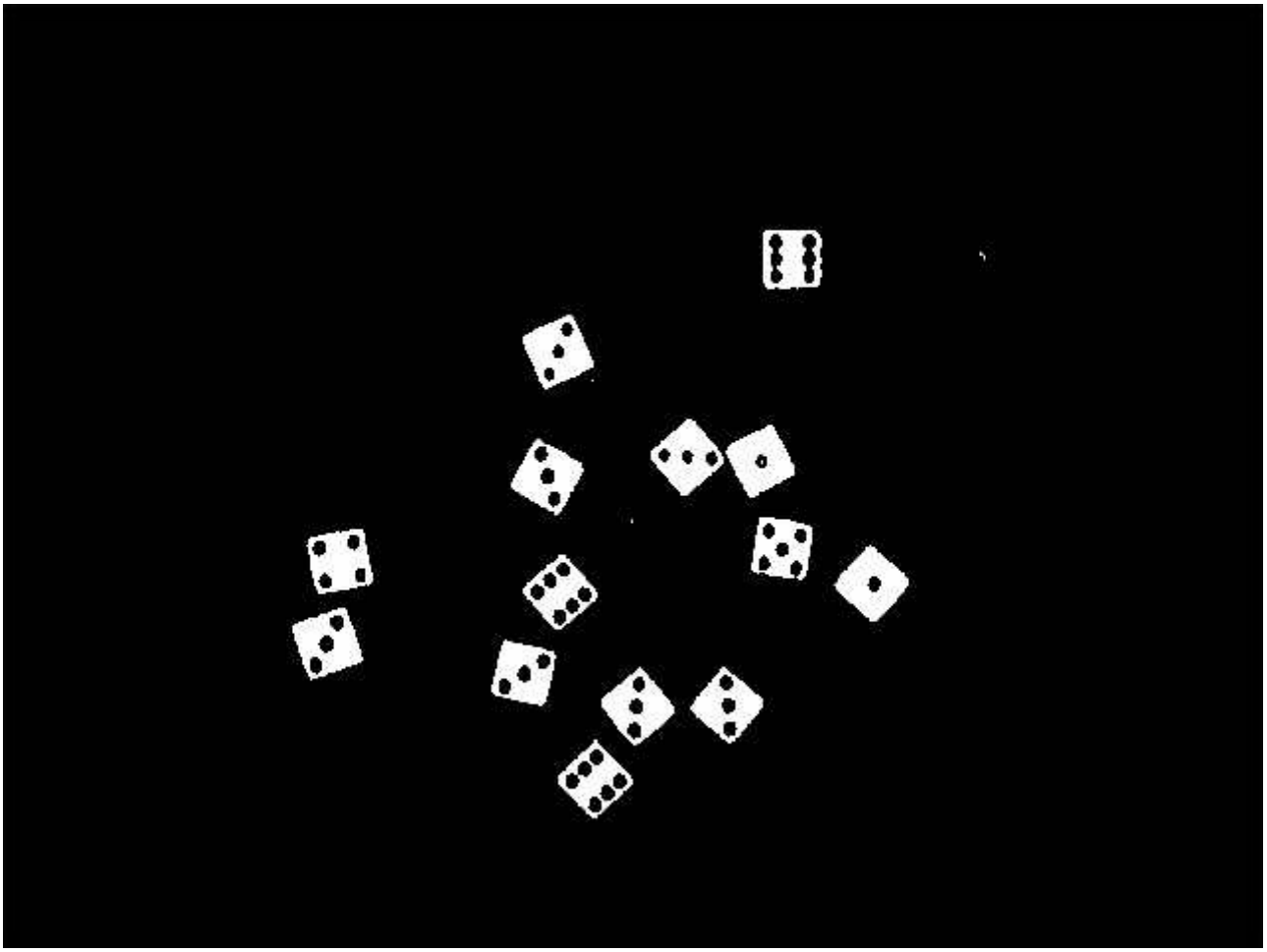


Imagem 2: Imagem resultante da conversão para preto e branco.

O segundo passo é a detecção dos contornos desta imagem a preto e branco. Como na imagem anterior já se eliminou tudo o que não é dados, fica-se apenas na presença dos contornos dos dados e das suas pintas. Esta detecção é conseguida convoluindo uma máscara de detecção de contornos com a imagem a preto e branco. A máscara utilizada é representada pela seguinte matriz:

$$\begin{bmatrix} 0 & -3 & 0 \\ -3 & 12 & -3 \\ 0 & -3 & 0 \end{bmatrix}$$

A convolução desta máscara com a imagem faz com que um dado pixel fique marcado como contorno se a sua vizinhança apresentar um elevado gradiante ou apenas num, ou em ambos os eixos coordenados.
Este passo já é mais lento que o anterior, mas mesmo assim ainda é bastante rápido. Para cada pixel, fazem-se 8 somas, 9 multiplicações e actualiza-se o valor dum outro vector. Continuando a assumir uma operação por ciclo de relógio do processador, obtemos 18 operações por pixel, o que, no total dá 18x307200 = 5529600 operações, ou seja, no exemplo do processador as 250MHz, consegue-se processar 45 imagens por segundo.

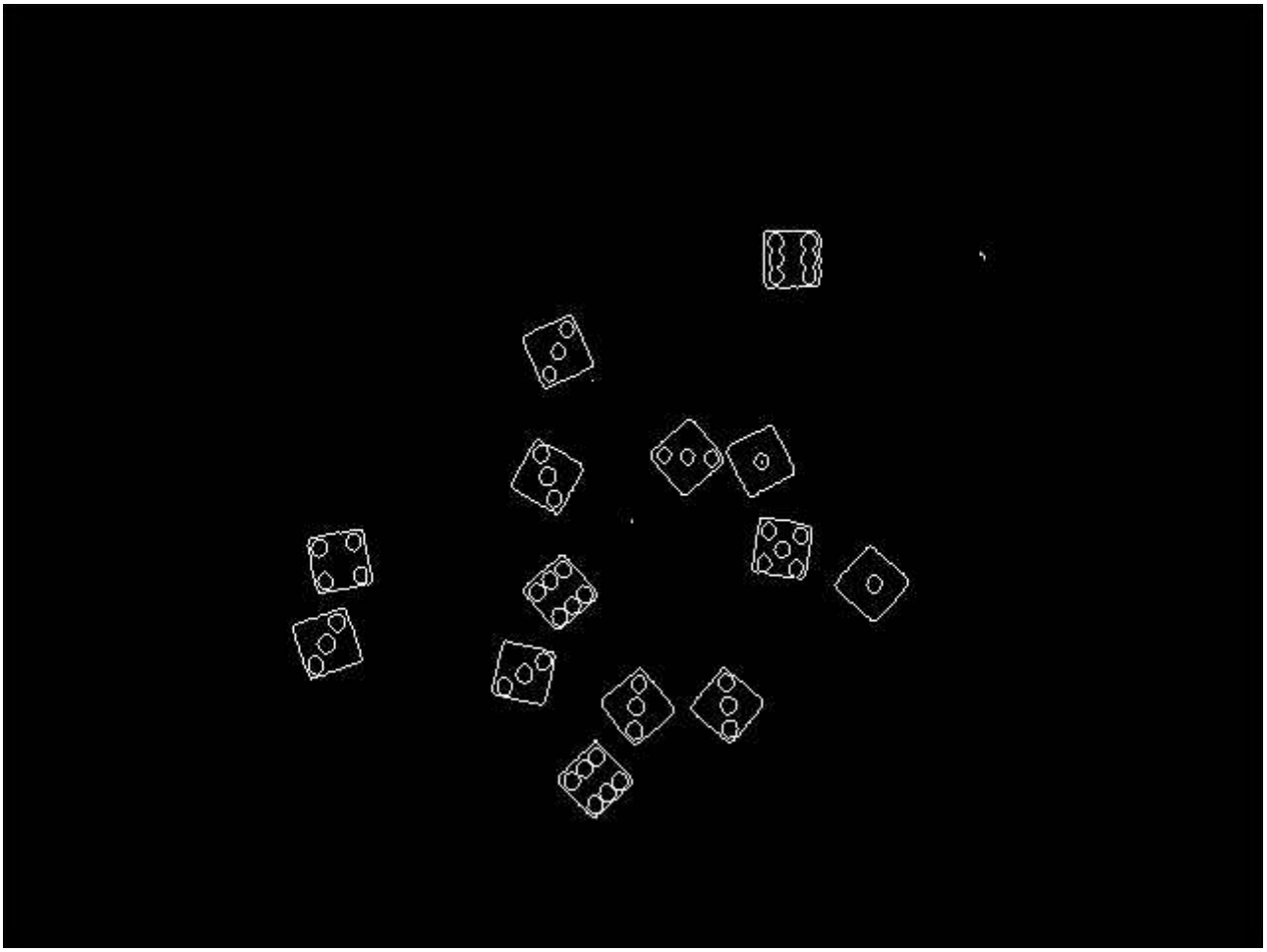


Imagem 3: Imagem resultante da detecção de contornos, na imagem a preto e branco.

2.2.1 - Detector de circunferências de Hough

Este detector consiste na criação duma nova imagem, com as mesmas dimensões que a imagem original, cujos pixeís servirão de contadores.
A imagem dos contornos é "varrida" e, quando se encontra um pixel de contorno, na imagem dos contadores, desenha-se uma circunferência, centrado nesse pixel de contorno e com o raio esperado (este raio é um parâmetro que se introduz *a priori*). Se estivermos na presença duma circunferência, o contador, no pixel central dessa circunferência, vai apresentar um valor relativamente alto, pois, por esse pixel, passam várias das circunferências desenhadas com base no contorno do círculo.
Para este método, por pixel, apenas se faz uma comparação, o que, para a imagem toda leva a 640 × 480 = 307200 operações por imagem. Assim, usando o processador a 250MHz, conseguem-se processar cerca de 814 imagens por segundo.
Obviamente, a velocidade real de processamento vai depender da quantidade de circunferências presentes na imagem, pois, por cada pixel de contorno, este método adiciona um em vários contadores (com raio 5, há cerca de 30 contadores a serem actualizados, por pixel de contorno).

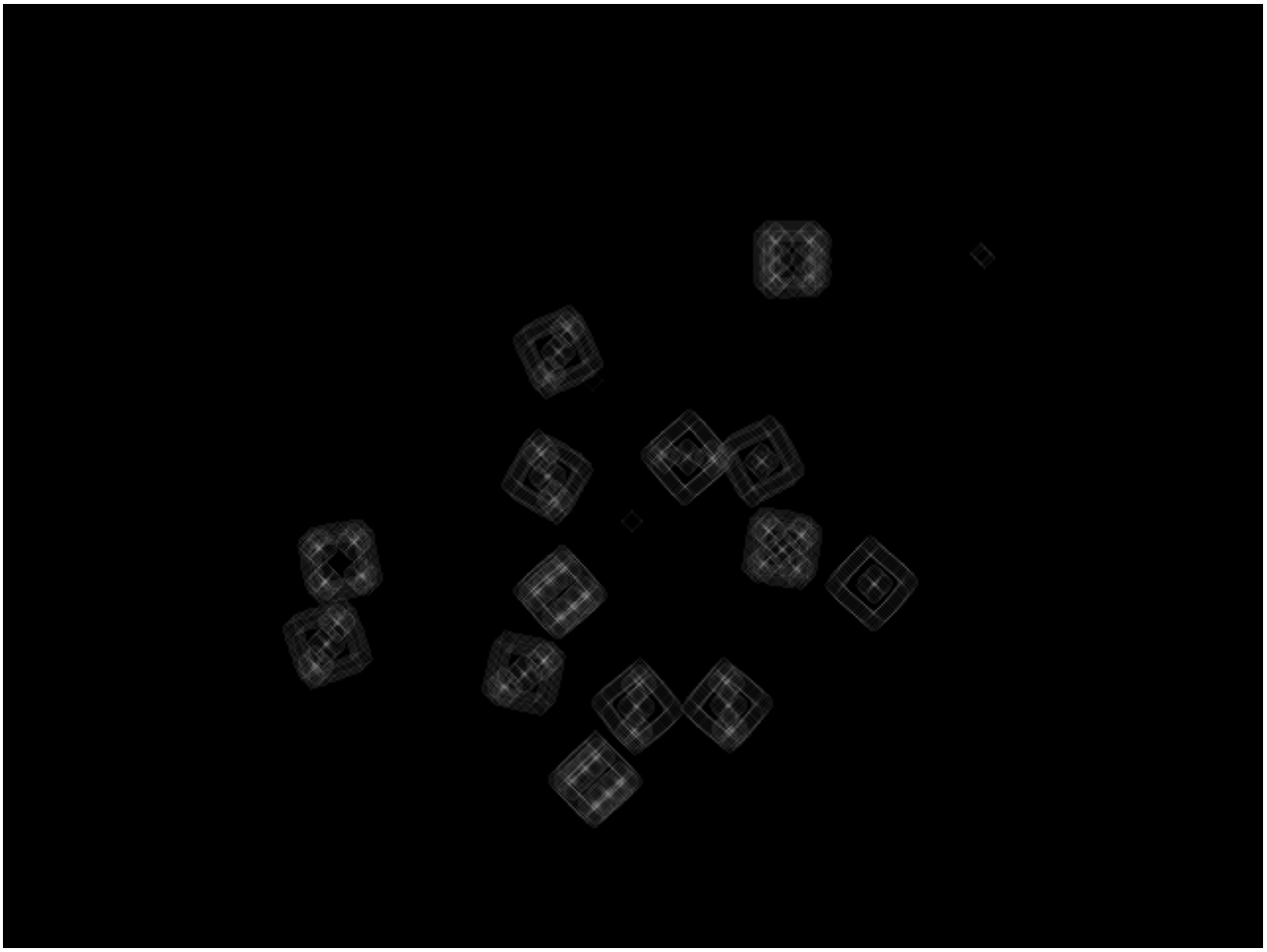


Imagem 4: Imagem resultante da aplicação da transformada de Hough. Os pixeis mais claros correspondem a centros de circunferências da imagem de contornos.

Nesta experiência, as imagens adquiridas apresentam circunferências de raio relativamente pequeno, o que faz com que, por vezes, o centro da circunferência não seja nenhum pixel. Isto leva a que não haja um pixel que

possa ser determinado como o centro da circunferência. Para esses casos (que são os mais frequentes), o algoritmo aqui desenvolvido vai um pouco mais além do algoritmo de Hough.
Quando a imagem dos contadores está completa, inicia-se a busca dos pixeis centrais. Quando um dado pixel apresenta um valor superior a um *threshold* (vamos-lhe chamar *Threshold1*), são feitas três médias: de quatro pixeis adjacentes, sendo o original o que está no canto superior esquerdo; de cinco pixeis adjacentes, em +, sendo o original o que está no centro; de cinco pixeis adjacentes, em x, sendo o original o que está no centro. Se a máxima destas três médias for superior a outro *threshold* (*Threshold2*), então está-se perante um muito bom candidato a centro. Neste momento, guarda-se a localização deste centro num vector que vai conter todos os centros detectados.
Mas, por vezes, encontram-se vários pixeis, todos eles perto do centro do mesmo círculo e a satisfazerem o critério do *Threshold2*. Para evitar que o mesmo centro seja contabilizado mais que uma vez, no vector que guarda a posição do centro já detectado, também é colocado o valor da média máxima. Assim, se um novo centro detectado está a uma distância inferior a um determinado valor (neste caso, usa-se o valor da estimativa do raio da circunferência), então comparam-se as médias e apenas fica registado, no vector, o centro que tiver maior média.

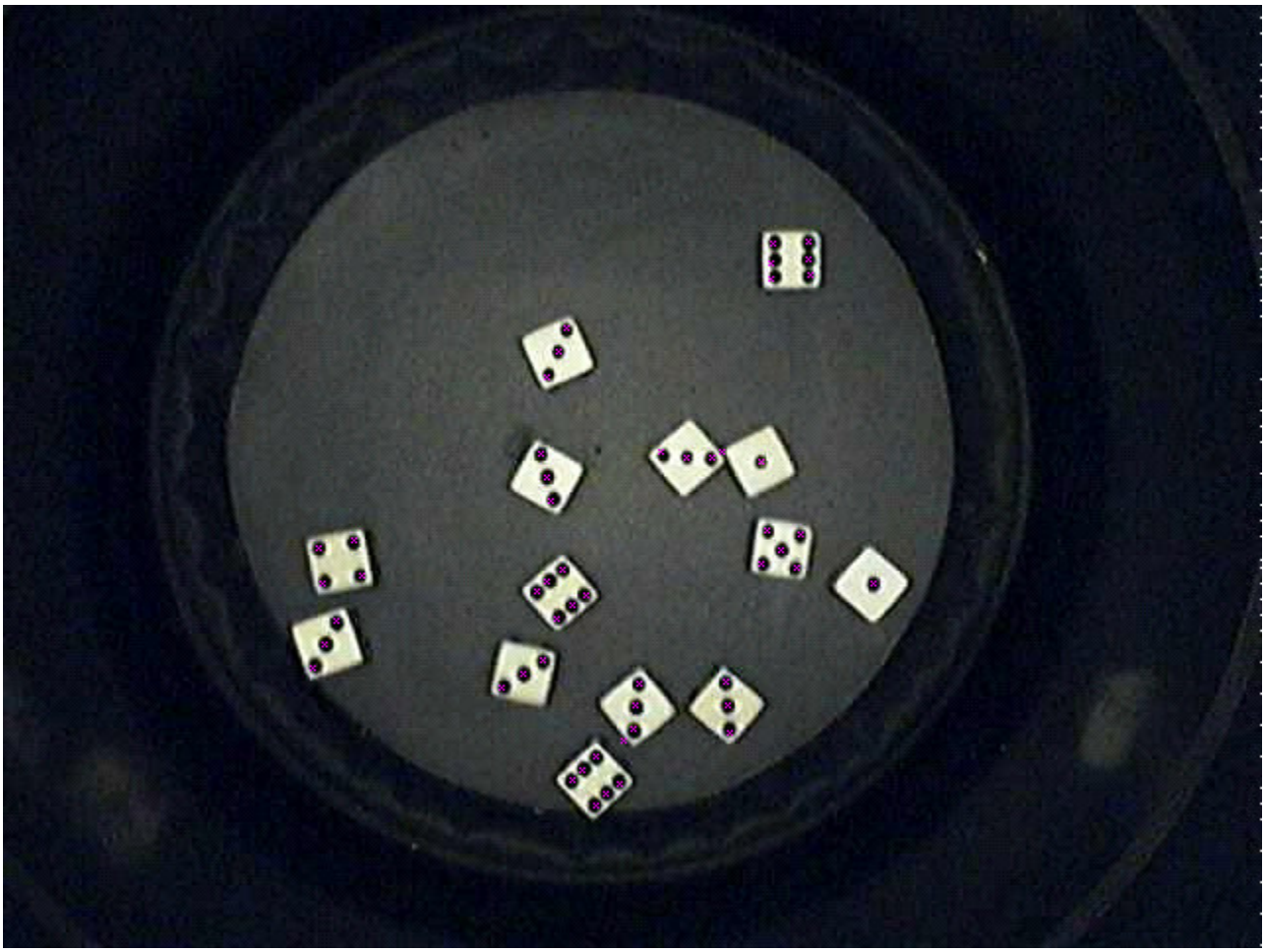


Imagem 5: Imagem resultante da primeira contagem, com base nos *Thresholds* definidos atrás.

Por outro lado, há algumas pintas que aparecem com o seu centro a branco, o que as iria eliminar, pois uma das medidas tomadas para eliminar más detecções foi a de eliminar pintas num pixel a branco (na imagem a preto e

branco). Para que estas pintas com o centro branco não sejam eliminadas, adicionou-se mais um *threshold* (*Threshold3*) – para valores da média indicada atrás superiores a este *Threshold3*, se a pinta estiver num pixel a branco, é aceite.

2.2.2 – Detector de circunferências por convolução

Este detector consiste em fazer uma adaptação da convolução entre uma matriz representativa duma pinta com o raio pretendido, subtraído de um valor para melhor ajuste, e a imagem a preto e branco. A adaptação mencionada prende-se com o facto de se se usasse uma convolução normal - ou seja, a matriz representativa duma pinta seria constituída por valores inteiros em que o branco seria, por exemplo, 1 e o preto 0 – as multiplicações por zero não seriam contabilizadas, o que levaria a vários erros de detecção. Assim, escolheu-se usar uma matriz em que os 1's são *booleans* a *true* e os zeros são a *false*. Deste modo, onde se faria uma multiplicação entre entradas da matriz e pixeis da imagem, faz-se uma comparação dupla: se a entrada for *true* e o pixel com um valor superior a zero ou a entrada a *false* e o pixel a zero.
Este processo torna-se bastante mais lento que o anteriormente descrito, pois, por cada pixel da imagem, tem-se, para o raio utilizado (5) e em média, $((raio-1)\times 2+1)^2 \times 2 = 162$ comparações, $((raio-1)\times 2+1)^2 = 81$ somas, uma divisão e uma comparação, ou seja, 162+81+1+1 = 245 operações por pixel. A convolução não é realizada nas periferias superior e esquerda da imagem, a uma distância da berma de, neste caso, $(5-1)\times 2+1 = 9$ pixeis, logo analisam-se $(640-9)\times(480-9) = 297201$ pixeis que, a 245 operações por pixel, leva a 72814245 operações por imagem. Usando o processador a 250MHz, processam-se cerca de 3,5 imagens por segundo.
Este método leva a resultados bastante bons, mas tal com o anterior também apresenta algumas más detecções, logo, dada a sua lentidão, foi abandonado para detectar pintas nos dados. Porém, ele mantém-se presente para o utilizador o usar, se bem que não influencia qualquer estatística.

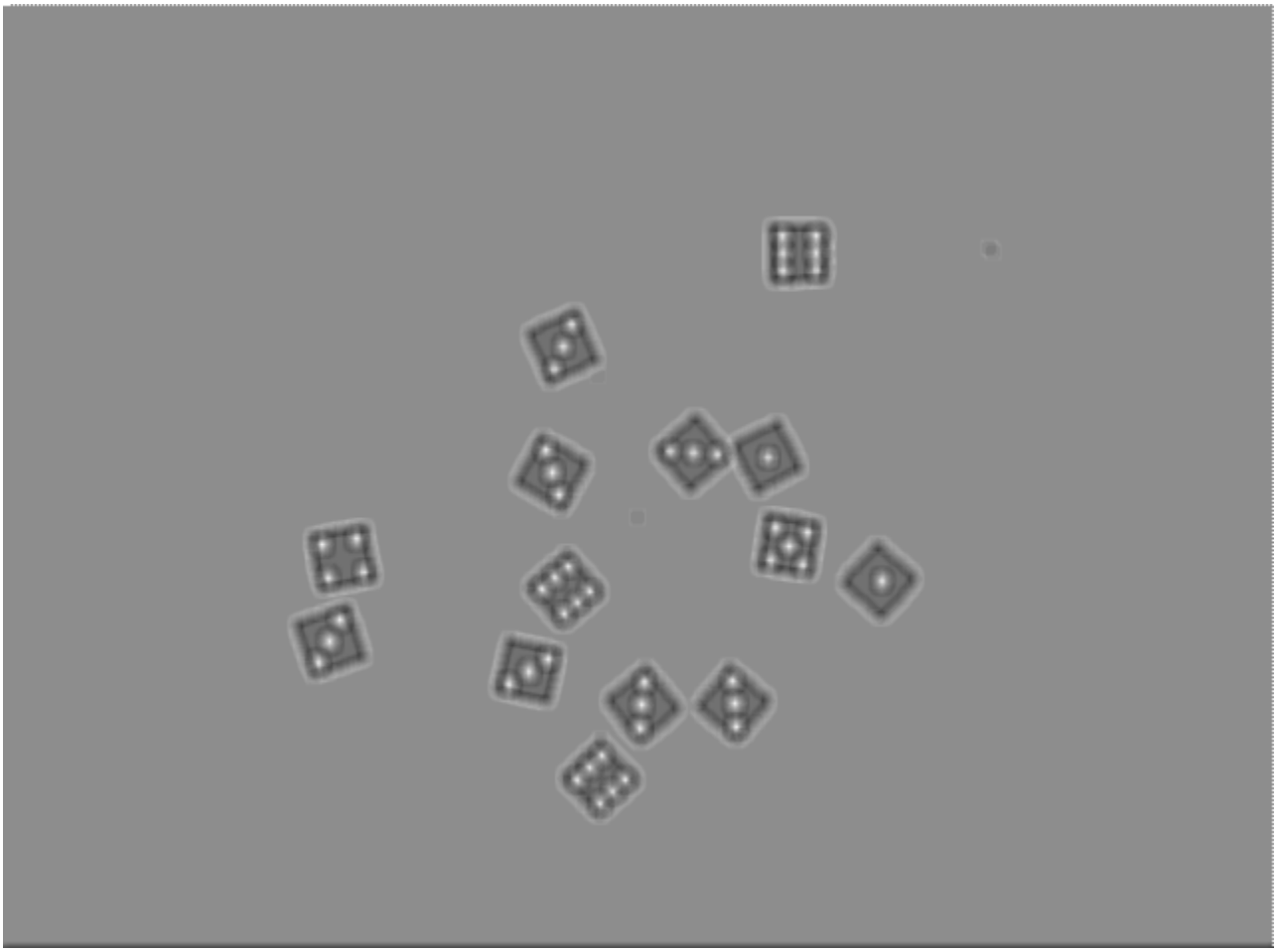


Imagem 6: Imagem obtida após a aplicação da convolução adaptada. Para a obtenção desta imagem, um Pentium4 a 2.52 GHz demora mais de 5 segundos. Nota-se que os pontos mais claros correspondem às pintas dos dados.

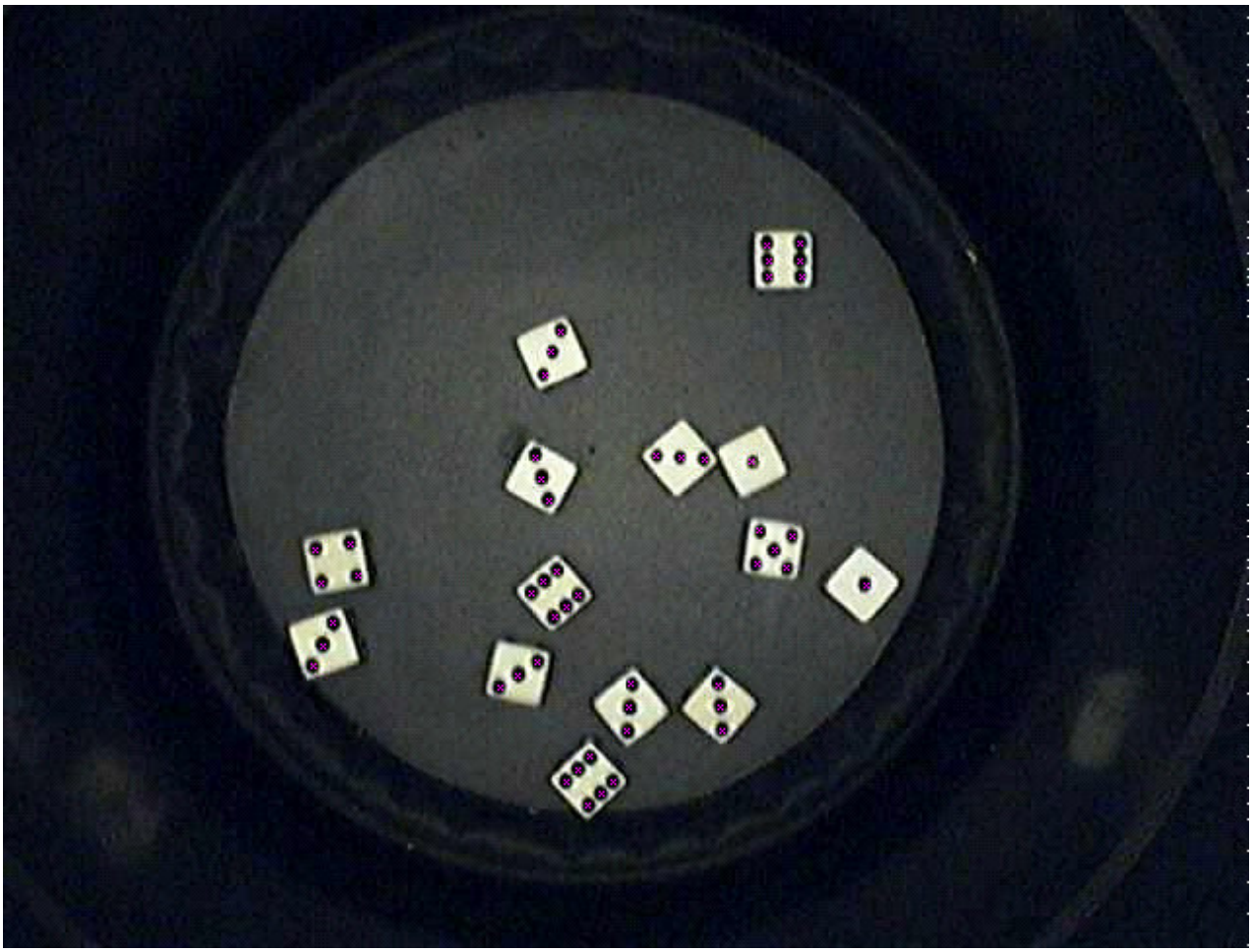


Imagem 7: Imagem obtida após a primeira contagem , usando um *Threshold* simples, de valor 190. Como se pode ver na imagem, este método conduz a resultados melhores que o método de Hough, mas a custo de mais tempo de processamento.

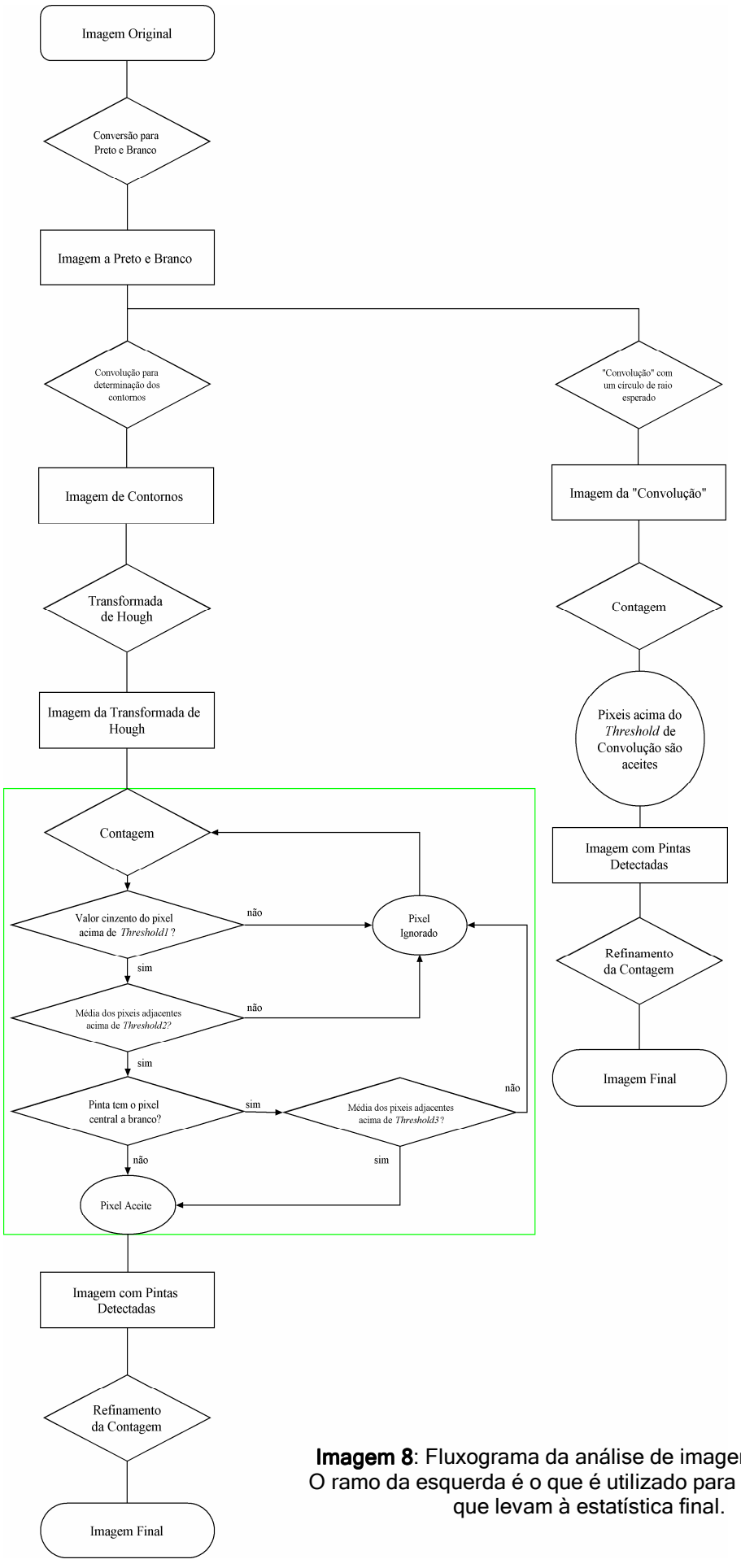


Imagem 8: Fluxograma da análise de imagem efectuada. O ramo da esquerda é o que é utilizado para as contagens que levam à estatística final.

2.2.3 – Refinamento da detecção

Após ser feita a primeira detecção de pintas, esta precisa de ser refinada, pois não é possível encontrar um conjunto de *thresholds* que apresente um resultado correcto para todas as imagens. Assim, os *thresholds* foram fixos de tal modo que todas as pintas são sempre detectadas. Infelizmente, esta atitude leva a que, na maioria das imagens, haja mais detecções do que pintas, levando à necessidade deste refinamento.
Esta é a parte do código mais especializada na detecção de dados de seis faces.
O primeiro passo deste refinamento é analisar as pintas detectadas e remover aquelas que, de certeza não são pintas reais. Para tal, usa-se a função "isSpot":

> -Função "isSpot":
> Partindo do pixel indicado como o centro, verificam-se os pixeis em seu redor, mas não todos, apenas os que estão na horizontal, na vertical e nas diagonais. Esta verificação é feita na imagem a preto e branco e, para cada um dos sentidos cria-se um contador que conta o número de pixeis a preto até se chegar a um a branco. Se estes contadores apresentarem valores dentro de certos limites, então a pinta não é considerada falsa e mantém-se para posterior análise; caso contrário, a pinta é considerada falsa. Este forma de analisar as pintas não pode ser assim tão simples, pois, por vezes, na conversão para preto e branco, as pintas não permanecem círculos fechados, mas abrem-se (acontece quando a pinta está muito perto da periferia do dado, como num 'dois', 'quatro', 'cinco' e 'seis'; também acontece entre pintas do 'seis'), levando a que os contadores apresentem valores fora dos limites determinados para se considerar pinta. Assim, foram consideradas várias condicionantes para eliminar pintas falsas, mas não excluir pintas verdadeiras, baseadas em algumas variáveis:
> -A função tem um parâmetro «isProbableSix» que é *boolean* e, quando *true*, os limites são um pouco grandes para que não sejam rejeitadas pintas pertencentes a um 'seis' cujas pintas se apresentem unidas e com a forma geral de duas minhocas; Assim, se *true*, a variável «maxCount» fica igual ao raio esperado das pintas multiplicado por 5.2, se *false*, fica raio multiplicado por 2.2.
> -A função tem um outro parâmetro «isProbableOne» que também é *boolean* e indica se o cluster que está a ser analisado é, muito provavelmente, um 'um'.
> -Criou-se uma variável que é a soma de todos os valores dos contadores, «sumCount».
> -Criou-se uma variável que indica a distância máxima entre os pontos que estão a ser considerados o final da pinta, «maxDiff».
> -Criou-se um contador que conta o número de vezes que os contadores de pixeis apresentam zero, «zeroCounter».
> -Criou-se um outro contador que conta o número de vezes que os contadores de pixeis ultrapassam o máximo estipulado, «maxCounter».
> -Criou-se também um contador que conta o número de vezes que os contadores de pixeis contam mais de zero e menos que o mínimo estipulado, «oneCounter».

Caso alguma das seguintes condicionantes se verificar, a pinta é considerada falsa:
-se o «zeroCounter» for maior que 2;
-se a soma do «zeroCounter» com o «maxCounter» for maior que 2;
-se o «zeroCounter» for maior ou igual a 1 e, ou o «oneCounter» for maior que 0, ou o «maxDiff» maior que o «maxCount»;
-se o «isProbableOne» for *true* e, ou o «zeroCounter» for maior que 1, ou o «maxCounter for maior que 1»;
-se o «zeroCounter» e o «maxCounter» são 0 e o «isProbableSiz» é *false* e o «maxDiff» é maior que o «maxCount»;
-se o «maxCounter» é maior ou igual a 2 e o «isProbableSix» é *false*;
-se o «maxCounter» é maior que 1 e o «maxDiff» é maior que o «maxCount»;
-se o «oneCounter» é maior que 6;
-se o «oneCounter» é maior que 1 e o «maxCounter» é maior que 0;
-se o «sumCount» é menor que um valor mínimo definido para esta variável (neste caso é 21) e, ou o «zeroCounter» é maior que 1, ou o «maxCounter» é maior que 0.
Se a pinta passar em todos estes testes, é considerada verdadeira. Está claro que é possível que uma falsa detecção passe em todos os testes, por isso há outras formas de remover pintas, noutras funções.
Esta função devolve um objecto *Vector* que tem várias componentes, sendo a primeira o resultado dos testes, ou seja uma variável *boolean*; e as seguintes são os contadores auxiliares: «zeroCounter», «oneCounter» e «maxCounter».

Cada pinta que não é removida por este exame passa a um outro que tem por objectivo que nenhum centro fique numa zona a branco (na imagem a preto e branco), sendo, os centros que estão no branco, mudados para o pixel a preto mais próximo. Isto permite uma análise mais generalista das pintas, sem que seja preciso criar código especial para estas (para além deste).
Após esta eliminação de pintas falsas, agrupam-se as pintas que restam em grupos (*clusters*). As pintas são agrupadas de modo a que nenhuma delas diste mais que um certo valor («maxClusterSize», neste caso este valor é 33, em unidades de pixeis) e, de seguida, faz-se uma análise cluster a cluster.
A cada cluster é associado um valor *boolean* (*true* ou *false*) que indica se o cluster já foi analisado e considerado como um dado real (*true*). Inicialmente, todos os clusters têm este valor a *false*.
De seguida, dá-se início a um *loop* que corre todos os clusters mais que uma vez se necessário, mas não mais que seis vezes, para não se tornar um loop infinito. Este loop acaba quando todos os clusters estão marcados como *true*. Se, após seis loops ainda houver clusters marcados como *false* e a sua quantidade for superior à quantidade de dados expecificados (neste caso, 14), removem-se tantos clusters a *false* quantos for preciso até chegar à contagem esperada (14). Se, mesmo assim ainda houver clusters a *false*, estes são mantidos.
O primeiro passo, dentro deste loop, é criar/redefinir um vector que contém os centros de todos os clusters (mais adiante estes centros serão precisos).
De seguida, se houver dois deste centros que distam menos que uma certa quantidade («maxClusterSize»/1.67) e estiverem ambos como *false* então estes dois clusters juntam-se num só. Isto prende-se com a necessidade de juntar clusters que têm pintas que pertencem ao mesmo dado.

De seguida procede-se à análise dos clusters. O primeiro, à partida, será o que tem a pinta situada o mais acima e à esquerda da imagem, sendo o critério vertical o mais ponderante, isto porque, em informática, as imagens são percorridas desde o canto superior esquerdo, na horizontal, até ao canto superior direito, passando para a linha de baixo e voltando a ir da esquerda para a direita e sempre assim até chegar, finalmente, ao canto inferior direito.
Os clusters sofrem uma análise diferente conforme o número de pintas presente. Assim, distinguem-se duas situações: clusters com uma só pinta e clusters com duas ou mais pintas.
-Clusters com uma só pinta:
A análise feita a estes clusters tem alguns componentes um pouco redundantes, mas servem para se fazer uma reverificação, dado que uma pinta sózinha pode pertencer a qualquer outro dado ou não pertencer a nenhum (falsa detecção) ou ser, de facto, um dado a 'um'.
Primeiro, verifica-se se o cluster é um dado a 'um'. Para tal usa-se a função "isOne":

> -Função "isOne":
> Esta função tem como parâmetros um vector de inteiros com o índice da imagem referente à pinta do cluster (para tratamento, a imagem é um vector de inteiros, com dimensão n*1, em que 'n' é o número total de pixeis da imagem e o sentido crescente é conforme o indicado atrás) e o tamanho máximo do cluster para referência e retorna um valor *boolean*.
> Procuram-se, em oito sentidos (dois horizontais, dois verticais e quatro diagonais) o pixel tal que o anterior era branco, ou seja, procura-se o final do dado. Esta busca parte desde o centro (indicado pelo índice da imagem) até à margem, mas não começa exactamente nesse centro, devido à existência de pintas com uma zona branca no centro. Assim, esta busca começa a 2 pixeis de distância do centro e termina quando ou detecta uma passagem de branco para preto, ou quando chega ao máximo que é definido pelo tamanho máximo do cluster. Para cada um dos sentidos obtém-se um valor que indica a distância a que está uma passagem de branco para preto, sendo comparados com valores de referência (no mínimo, tamanho máximo do cluster a dividir por 3.3 , no máximo a mesma variável a dividir por 1.5). Esta comparação é feita no sentido dos ponteiros do relógio, começando no valor do sentido para cima e, se houver mais de duas abaixo do mínimo e que haja pelo menos duas consecutivas nesta situação, o cluster é considerado como não sendo um 'um'; se houver mais de três abaixo do mínimo e nenhuma consecutiva nesta situação, também não é considerado um 'um'; por fim, se houver algum valor acima do máximo o cluster também não é considerado um 'um'.

Se a função "isOne" retornar um *true*, então é chamada a função "checkDie1" que vai ser a responsável por considerar o cluster como *true*.

> -Função "checkDie1", parte 1 - «isProbableOne» = *true*:
> Esta função tem uma série de parâmetros que servem apenas para transporte de variáveis, excepto um, «isProbableOne»: este parâmetro é *boolean* e indica que este cluster é, muito provavelmente, um 'um', caso seja *true*.

Esta função faz umas verificações iniciais, recorrendo à função "isSpot", inicialmente, num modo relaxado, ou seja com o «isProbableSix» a *true* e o «isProbableOne» a *false*. Caso tenha um resultado *false*, repete-se a chamada a "isSpot", mas com o «isProbableSix» a *false* e o «isProbableOne» a *true*. Caso este resultado também seja *false* e, ou o «zeroCounter» desta última chamada seja maior que 1, ou o «oneCounter» seja maior que 2, ou o «maxCounter» seja maior que 1, a pinta é considerada falsa e é eliminada. Assim, fica-se com um cluster vazio que é removido da lista.
Se a pinta passar nestas verificações, passamos ao próximo passo. Este passo tem duas vertentes, mas aqui só se vai tratar da vertente «isProbableOne» a *true*.
Aqui, faz-se uma chamada a "isOne" e, caso esta função retorne *false*, a pinta é considerada falsa e é removida, assim como o cluster que a continha; se retornar *true*, a pinta é considerada verdadeira, tal como o cluster que passa a ser *true*, ou seja fica considerado como um dado.

Se a função "isOne" retornar *false*, também é chamada a função "checkDie1", mas com o «isProbableOne» a *false*:

-Função "checkDie1", parte 2 - «isProbableOne» = *false*:
As verificações iniciais, neste caso são exactamente as mesmas que no caso anteriormente considerado.
Como, agora, se tem a variável «isProbableOne» a *false*, a função vai verificar se esta pinta está suficientemente perto dum outro cluster, para que lhe possa pertencer. Para isso, determina o cluster que lhe está mais próximo e, se este estiver a menos duma dada distância e ainda não estiver a *true*, então a pinta passa para esse cluster. Caso o cluster para onde a pinta vai passar apenas tenha uma pinta, então, antes de fazer a transferência, verifica-se se o resultado é um 'dois', recorrendo à função "isTwo" (descrita mais a baixo), com o parâmetro «isRelaxed» a *false*.
Se o cluster se mantiver com a pinta e houver algum cluster a uma distância inferior a «maxClusterSize», então verifica-se se é possível ir buscar uma pinta a esse cluster. Para tal verifica-se a distância entre casa pinta desse cluster com a pinta do cluster presente e, se essa distância for inferior a «maxClusterSize»*0.795 e a função "isTwo" retorne *true* (com o parâmetro "isRelaxed" a *false*), então a pinta do outro cluster é transferida parao cluster corrente e este fica com duas pintas. Como está no "CheckDie1", a função retorna os clusters após esta transferência e mais um parâmetro a *false*, indicando que este cluster ainda não foi totalmente analisado. Assim, o programa volta a analisar o cluster que, desta vez, já tem duas pintas.
Por vezes, o cluster cujo centro está mais próximo da pinta que se está a analisar não é o que contém a(s) pinta(s) do dado ao qual pertence esta pinta. Assim, procura-se, em todos os outros clusters que estejam a *false*, uma pinta tal que, quando confrontada com a função "isTwo", com o parâmetro «isRelaxed» a *false*, retorne *true*. Se isso acontecer, essa pinta é transferida para o cluster corrente.

Se esta chamada a "checkDie1" retornar o *boolean* a *true*, faz-se uma nova chamada a esta função, desta vez, com o parâmetro «isProbableOne» a *true*.

-Clusters com mais que uma pinta:

Para analisar clusters com mais que uma pinta, recorre-se à função "checkDie":

-Função "checkDie":

Nesta função, os parâmetros são apenas para transferência de variáveis.

Primeiro, determina-se o "centro de massa" do cluster e cria-se um vector com as distâncias de cada pinta a este "centro de massa". Este vector é passado a uma função que vai determinar se o número de pintas é compatível com a sua localização no cluster, "checkSpotCount":

-Função "checkSpotCount":

Esta função tem como parâmetros um vector com as distâncias de cada pinta ao "centro de massa" do cluster, o número de pintas, um vector com os índices das pintas na imagem e o «maxClusterSize».

Se o número de pintas for 1, a função chama a função "isOne" e retorna o que esta retornar;

se o número de pintas for 2, chama a "isTwo", com o parâmetro «isRelaxed» a *false* e retorna o que esta retornar;

se o número de pintas for 3, chama a "isThree" e retorna o que esta retornar;

se o número de pintas for 4, chama a "isFour" e retorna o que esta retornar;

se o número de pintas for 5, chama a "isFive" e retorna o que esta retornar;

se o número de pintas for 6, chama a "isSix" e retorna o que esta retornar;

se o número de pintas for superior a 6, retorna *false*.

-Função "isTwo":

Esta função tem como parâmetros um vector com as distâncias de cada pinta ao "centro de massa" do cluster, um vector com os índices das pintas na imagem e um *boolean*, «isRelaxed», que, quando *true* indica que as variáveis «whitecounterNumMin» e «minWhitePixelCounter» são 0 e «whitecounternumMax» e «maxWhitePixelCounter» são 1000; quando o «isRelaxed» é *false*, fica «whitecounterNumMin» e «whiteCounterNumMax» a 1, «minWhitePixelCounter» a 15 e «maxWhitePixelCounter» a 33.

Definem-se duas constantes «minDist» e «maxDist» que indicamos valores mínimo e máximo que as entradas do vector com as distâncias têm que ter.

Primeiro, cria-se um vector de inteiros, no qual é colocado o output da função "countWhitePatches":

-Função "countWhitePatches":

Esta função tem como parâmetros dois inteiros, um indica o índice do primeiro pixel, na imagem, e o outro o índice do segundo pixel.

De seguida, a função dedica-se a contar o número de pixeis entre o primeiro e o segundo que estão a branco e o número de sequências de pixeis a branco. (por exemplo,

num 'dois' está-se à espera que entre as duas pintas apenas haja uma sequência de pixeis brancos).
Esta função retorna um vector de inteiros com estes dois contadores.

Se todas as entradas do vector das distâncias estiverem entre os «minDist» e «maxDist» e se o número de sequências de pixeis brancos estiver entre «whiteCountenumMin» e «whitecountenumMax» e o número de pixeis contados a branco estiverem entre «minWhitePixelCounter» e «maxWhitePixelCounter», então esta função retorna *true*; caso contrário, retorna *false*.

-Função "isThree":
Esta função tem como parâmetros um vector com as distâncias de cada pinta ao "centro de massa" do cluster e um vector com os índices das pintas na imagem.
Determina quais das três pintas são as duas periféricas e, usando a função "countWhitePatches" verifica se, entre estas duas pintas, há uma zona a preto e se a contagem de pixeis a branco nesse percurso está de acordo com o padrão esperado.
Para além disto, verifica se as distâncias das pintas ao centro de massa coincidem com o padrão de um três – uma pinta no centro e as outras duas quase equidistantes do centro e dentro de certos valores determinados pela experiência.
Se todos estes teste derem resultado favorável, então a função devolve *true*, caso contrário, devolve *false*.

-Função "isFour":
Esta função tem como parâmetros um vector com as distâncias de cada pinta ao "centro de massa" do cluster e um vector com os índices das pintas na imagem.
Primeiro, verifica se as distâncias das pintas ao centro estão dentro de valores determinados experimentalmente. Se esta verificação falhar, a função retorna *false*, se não falhar, faz um outro pito de verificação. Esta segunda verificação consiste em determinar as distâncias de cada pinta às outras do cluster. Espera-se que duas destas distâncias sejam semelhantes e a outra seja superior. Se esta verificação falhar, retorna *false*; se não falhar retorna *true*.

-Função "isFive":
Esta função tem um único parâmetro, um vector com as distâncias de cada pinta ao "centro de massa" do cluster.
Primeiro, verifica se existe uma e uma só pinta marcada no centro (ou muito perto) do cluster. Se não existir tal pinta, retorna imediatamente *false*. Se existir, faz uma verificação em tudo semelhante à primeira verificação feita na função "isFour", para confirmar qeu se trata de algo muito próximo do cinco. Tal como as anteriores, se as pintas do cluster não estiverem de acordo com o esperado, retorna *false*, caso contrário, retorna *true*.

-Função “isSix”:
Esta função tem como parâmetros um vector com as distâncias de cada pinta ao “centro de massa” do cluster e um vector com os índices das pintas na imagem.
Primeiro, verifica se as distâncias presentes no vector estão dentro de valores determinados experimentalmente. Caso estejam, determina quais são as duas pintas centrais do seis e verifica se há duas próximas de cada uma delas. Deste modo, sabe-se que se está perante um seis, dado se ter verificado a partir de dois pontos de vista diferentes. Tal como nas funções anteriores, esta função retorna *false* se algum dos testes falhar e retorna *true* caso contrário.

De seguida, verifica se todas as pintas são potencialmente pintas reais, recorrendo à função “isSpot”, com o parâmetro “isProbableSix” a *true* se o cluster tiver seis ou mais detecções e *false* se tiver menos de seis e o parâmetro “isProbableOne” a *true* se o cluster tiver apenas uma detecção e *false*, caso contrário.
Caso algumas das duas verificações atrás referidas (“checkSpotCount” e “isSpot”) retornar *false*, entra num ciclo *while* que, enquanto não acontecer que, numa verificação posterior e da mesma natureza, retorne valor *true*. Para além disto, para salvaguardar que não se fica num ciclo infinito, ao fim de seis ciclos, termina. Usa-se este valor máximo de seis, porque cada dado tem até seis pintas.
Dentro deste ciclo, se o cluster tiver apenas uma detecção (possível, após uma ou mais iterações do ciclo *while*), tem um tratamento igual àquele que tem um cluster com apenas uma detecção, no início.
Se o cluster tiver duas detecções, verifica-se se alguma das detecções pode pertencer a um seis, procurando detecções que estejam bastante próximas destas duas (esta proximidade apenas se verifica num seis). Com este conhecimento, usa-se a função “isSpot” com o parâmetro “isProbableSix” de acordo com o determinado anteriormente, para cada uma das detecções. Se esta função retornar *false* a pinta correspondente a este retorno é eliminada. Se, depois destas verificações, o cluster continuar com duas detecções, determina-se se estão a uma distância superior à esperada para um dois e, em caso afirmativo, assume que não se trata de um dois, mas sim de dois uns e, como tal, separa o cluster em dois. Ao que sobra, faz um teste para verificar se é um um, tal como anteriormente se fazia para testar clusters só com uma detecção.
De seguida, aplica-se a função “checkSpotCount” e, caso esta retorne *true*, assume-se que o cluster presente corresponde a um dado (com uma ou duas pintas) e sai-se da função, retornando um objecto *Vector*, cujas componentes são um outro objecto, “vDie” (um vector de *Vector*'s que tem guardados os índices das detecções, correspondentes a cada cluster, ou seja, em cada entrada deste vector, está um *Vector* de interios com estes índices, correspondente a um cluster; a última entrada deste *Vector* tem um valor *boolean* que indica se o cluster está identificado com um dado ou não); neste *Vector* que é retornado, também está um valor *boolean* que indica para passar ao próximo índice dos clusters, ou seja, para seguir para o próximo cluster (este *boolean* torna-se necessário, pois é possível que aconteça que o cluster

analisado fique sem qualquer detecção e, nesse caso, este cluster é eliminado e o próximo tem o mesmo índice que o anterior).
Se o cluster continua com duas detecções e a função "checkSpotCount" retorna *false*, então este cluster é dividido em dois.
Se o cluster não tiver nenhuma detecção, então é eliminado e a função retorna o *Vector* com o "vDie" e o valor *boolean* a *false*.

O resto desta função é aplicado a todos os outros casos e quando estes casos anteriores não retornam, ou seja, ainda têm alguns testes a fazer.
Primeiro, apllica-se a função "checkSpots":

-Função "checkSpots":

Esta função tem como parâmetros o "vDie", o índice do cluster que se está a analisar, um vector com as posições, no eixo dos xx, das detecções, outro vector com as posições no eixo dos yy, um vector com os índices de imagem das detecções, o valor do raio esperado das pintas (em pixeis) e o valor do tamanho máximo esperado para a diagonal do dado (pinta a pinta, em pixeis).
Caso se esteja perante um cluster com três detecções, esta função tem umas verificações extra:

Recorrendo à função "countWhitePatches", determina se, entre as duas detecções extremas existem duas zonas a branco. Se existirem menos de duas destas zonas, procura-se a detecção com maior probabilidade de não ser pinta, recorrendo à função "isSpot" e esta é transferida para um novo cluster. Após esta transferência, verifica-se se o cluster resultante tem o padrão correspondente ao seu número de detecções (usando a função "checkSpotCount") e, em caso afirmativo, a função retorna um *Vector* com 5 componentes: "vDie", o vector de xx, o vector de yy, o vector com os índices de imagem das detecções e um valor *boolean* a indicar se este cluster está definido.
Se o padrão não corresponder, a função continua pelo mesmo algoritmo que é usado para todas as outras quantidades de pintas.
Para cada detecção, verifica se pode pertencer a um seis – se o cluster tiver seis ou mais detecções, muito provavelmente, a pinta vai pertencer a um seis; se a detecção mais próxima desta for compatível com a proximidade de pintas num seis, então assume-se que pode pertencer a um seis – definindo-se, assim, a variável "isProbableSix" que vai ser fornecida à função "isSpot"; o parâmetro "isProbableOne" é *true* se o cluster tiver apenas uma detecção e *false* caso contrário. Se esta função retornar *false*, esta detecção é eliminada e verifica-se se o resultado é compatível com um dado, com a função "checkSpotCount". Se fôr, então a função retorna o *Vector* indicado atrás, com o valor *boolean* a *true*; se não fôr, verifica a próxima detecção. Tendo chegado ao fim de todas as detecções e a "checkSpotCount" continua sem considerar o cluster compatível a um dado, a função retorna o *Vector*, mas com o valor *boolean* a false, indicando que este cluster ainda precisa de mais tratamento para se chegar a alguma conclusão.

A análise seguinte é feita usando a função “checkDoubleSpots” que vai verificar se há duas detecções na mesma pinta do dado:

-Função “checkDoubleSpots”:

Esta função tem como parâmetros o “vDie”, o índice do cluster que se está a analisar, um vector com as posições, no eixo dos xx, das detecções, outro vector com as posições no eixo dos yy, um vector com os índices de imagem das detecções, o valor do raio esperado das pintas (em pixeis) e o valor do tamanho máximo esperado para a diagonal do dado (pinta a pinta, em pixeis).

Para cada detecção, verifica, com a função “isSpot” se se trata, provavelmente, duma pinta (“isProbableSix” a *true* se o número de detecções no cluster fôr igual ou superior a seis e o “isProbableOne” a *true* se o número de detecções no cluster fôr um) e, caso se trate, verifica a distância que vai desde ela até às outras detecções do cluster. Se houver alguma detecção que fica a uma distância inferior ou igual a um raio mais um pixel, elimina-se a segunda detecção.

Por cada eliminação, verifica-se se o cluster resultante já é compatível com algum dado (usando a função “checkSpotCount”) e, em caso afirmativo, retorna um *Vector* igual ao retornado pela função anterior e com o valor *boolean* a *true*. Se chegar ao fim e não tiver eliminado nenhuma detecção ou a “checkSpotCount” nunca tenha retornado *true*, este *Vector* é retonado com o *boolean* a *false*.

A análise seguinte é feita usando a função “checkIfSpotsBelongToAnother” que verifica se alguma detecção pode pertencer a um outro cluster e transfere-a, em caso afirmativo:

-Função “checkIfSpotsBelongToAnother”:

Esta função tem como parâmetros o “vDie”, o índice do cluster que se está a analisar, um vector com as posições, no eixo dos xx, das detecções, outro vector com as posições no eixo dos yy, um vector com os índices de imagem das detecções, o valor do raio esperado das pintas (em pixeis) e o valor do tamanho máximo esperado para a diagonal do dado (pinta a pinta, em pixeis).

Para cada detecção, guarda-se o índice do cluster que lhe está mais perto e a distância. De seguida, define-se uma grandeza, “maxmaxDist”, que é a distância máxima para a qual se considera que a detecção faz parte do outro cluster. Se o cluster presente tiver uma ou duas detecções esta variável é maior do que se o cluster tiver mais detecções, porque, com uma detecção, pretende-se verificar se pode ir buscar uma outra detecção e ficar um dois, com duas detecções, este cluster pode fazer parte dum três ou dum cinco.

Se a distância da detecção ao cluster estiver abaixo de “maxmaxDist”, então, em princípio, vai-se mudar a detecção para o outro cluster, mas, se o outro cluster já estiver definido como compatível com a forma dum dado, primeiro testa-se se esse cluster com a nova detecção continua a ser compatível com um dado. Se sim, então efectua-se a transferência, se não, fica na

mesma. Se o outro cluster não estiver definido como compatível com a forma dum dado e tiver apenas uma detecção, verifica-se se o resultado da transferência da detecção para esse cluster resulta num dois e apenas se faz a transferência se fôr, de facto, um dois. Para qualqquer outra quantidade de detecções no cluster, faz-se a transferência.

De seguida, fazem-se algumas verificações para verificar se o cluster resultante ficou com um padrão compatível com algum dado.

Esta função retorna um *Vector* igual ao da função anterior com a variável *boolean* com o mesmo significado.

Se, até este momento, o cluster ainda não tiver sido marcado como compatível com algum dado, verifica-se se algum subconjunto de detecções, dentro do cluster, tem um padrão compatível com algum dado, sempre verificando, primeiro, para os dados com mais pintas e, no final, para os dados com menos pintas. Assim, apanha-se, por exemplo, um cinco, antes de se apanhar um três, dado que um três é um subconjunto dum cinco.

Esta verificação só é feita até uma diferença entre o número de detecções do cluster e o dados a ser testado ser de quatro. Mais que quatro é uma grande diferença e muito pouco provável.

Apesar desta precaução, se o cluster tiver, inicialmente, 4 detecções e, neste passo, passar a ter 3, este 3 pode muito bem fazer parte dum cinco; ou se passar a ter duas, pode fazer parte dum quatro ou dum seis. Assim, após este passo, para o caso de se ter um cluster com três ou duas detecções e estiver difinido como compatível com um dado, verifica-se se há detecções numa grande proximidade dele, de tal modo que continue a ser compatível com um dado.

De seguida, faz-se a última verificação com a função "checkSpotsFromOthers" que procura, nos outros clusters, detecções que possam pertencer a este, dada a sua proximidade.

-Função "checkSpotsFromOthers":

Esta função tem como parâmetros o "vDie", o índice do cluster que se está a analisar, um vector com as posições, no eixo dos xx, das detecções, outro vector com as posições no eixo dos yy, um vector com os índices de imagem das detecções, o valor do raio esperado das pintas (em pixeis) e o valor do tamanho máximo esperado para a diagonal do dado (pinta a pinta, em pixeis).

Esta função, por construção tem tendência a acumular bastantes detecções no mesmo cluster, então, tem um controlo do número de detcções presentes no cluster, de tal modo que sai da função se e quando este número é igual ou superior a 6.

Primeiro determina o cluster que está mais próximo de cada detecção e, se este cluster já estiver definido como compatível com algum dado, verifica se, com a adição desta nova detecção, continua a ser compatível com algum dado. Se sim, trata da transferência da detecção. Se o cluster não estiver definido como compatível com algum dado, faz a transferência imediatamente. Depois desta transferência, se o cluster onde estava a detecção ficar vazio, é eliminado da listagem. Se o cluster presente ficar

com um padrão compatível com algum dado, a função retorna um *Vector* igual ao da função descrita atrás, com o valor *boolean* a *true*. Se chegar ao fim da função sem que o cluster esteja identificado com nenhum dado, esse valor *boolean* é retornado a *false*.

Chegada a este ponto, a função faz uma nova chamada à função "checkSpotCount", para actualizar a variável *boolean* "isSpotCountOK" que, caso seja *false* vai levar a que se reiniciem as várias verificações feitas nesta função, mas esse reiniciar só é feito, no máximo, 6 vezes, para não ficar num *loop* infinito.
Se, depois deste *loop* ser processado as 6 vezes, ainda não houver sucesso, o cluster é completamente separado, ou seja, cada detecção é colocada em seu cluster novo e verifica-se se a detecção que fica no cluster é compatível com um 'um', de acordo com o procedimento indicado para um cluster que tem uma detecção no início.
Se a "isSpotCountOK" ficar a *true*, então o cluster fica marcado como compatível com algum dado.
Esta função retorna um *Vector* com a "vDie" e uma *boolean*, que, chegada a este ponto, é colocada a *true*, indicando que a função que chamou esta pode seguir para o próximo cluster.

Após a chamada a "checkDie", que é onde está o cerne do refinamento da contagem, verifica-se se há algum cluster a uma distância bastante próxima do cluster actual, o que faz com que se junte os dois clusters. Deste modo, se, por exemplo, ficarmos com um cluster com uma detecção e outro com quatro detecções, mas têm os seus centros bastante próximos, então esses dois cluster juntam-se para, posteriormente ser verificado se se trata dum cinco.
Finalmente, após todos os clusters serem analisados, se houver algum que ainda não esteja identificado com um dado, este é eliminado, mas apenas se o número de clusters fôr superior ao numero de dados esperado (este parâmetro faz parte das configurações). Se o número de clusters presente não ultrapassar o número de dados esperado e houver algum cluster que ainda não esteja identificado, este é mantido.
Seguidamente, constroi-se a imagem final com circunferências de cor magenta de centro no centro dos clusters e raio a metade da variável "maxClusterSize" (que faz parte das configurações) e desenham-se cruzes centradas em cada detecção. Cada cluster fica com cruzes de cor diferente, para se diferenciarem as detecções que ficam em cada cluster.

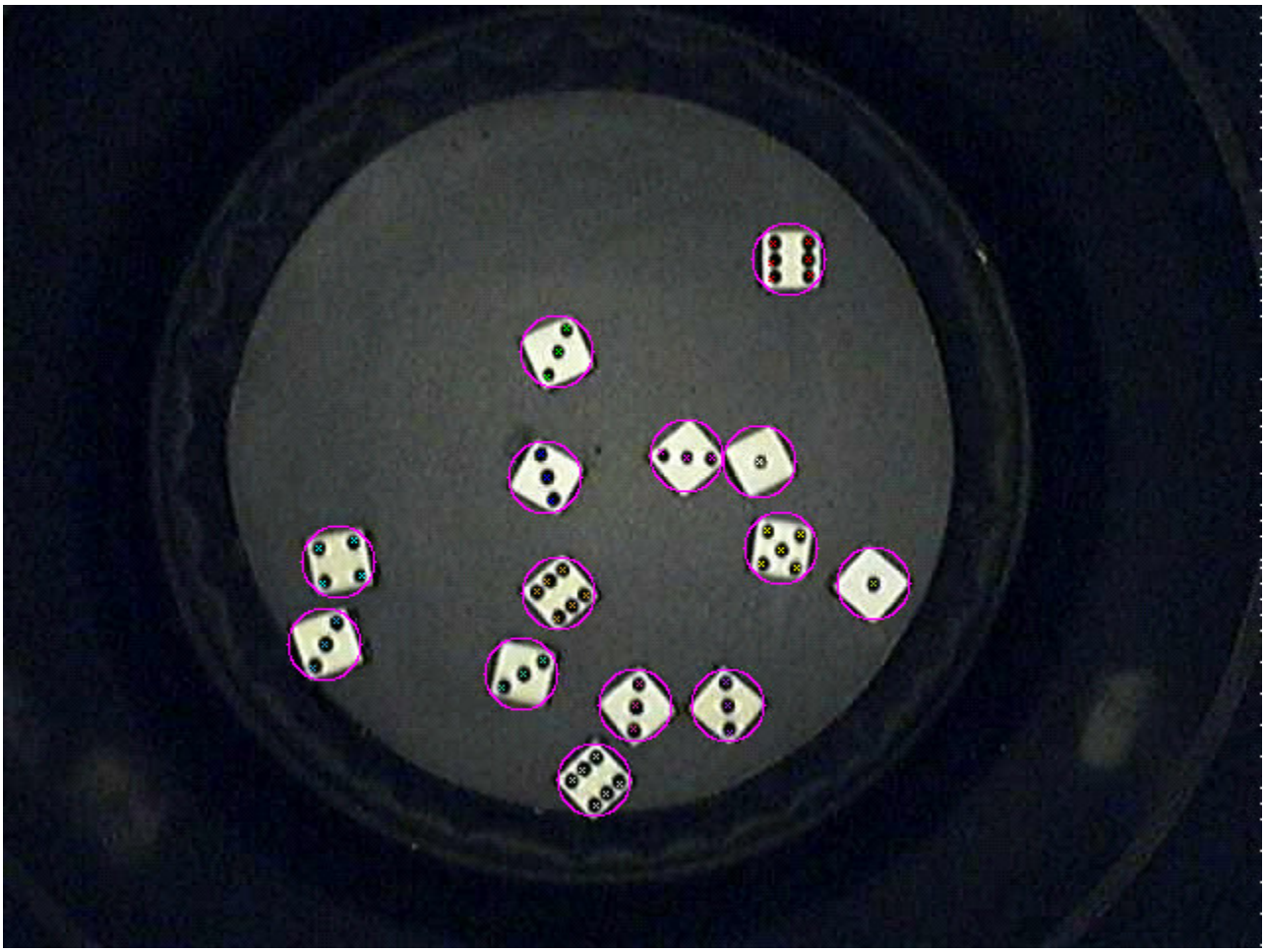


Imagem 9: Imagem obtida após o refinamento da contagem. Apresenta cada dado identificado através da circunferência de côr magenta de centro no "centro de massa" das detecções do cluster. Cada cluster tem as suas detecções marcadas a uma cor diferente, para se verificar que o dado está bem identificado.

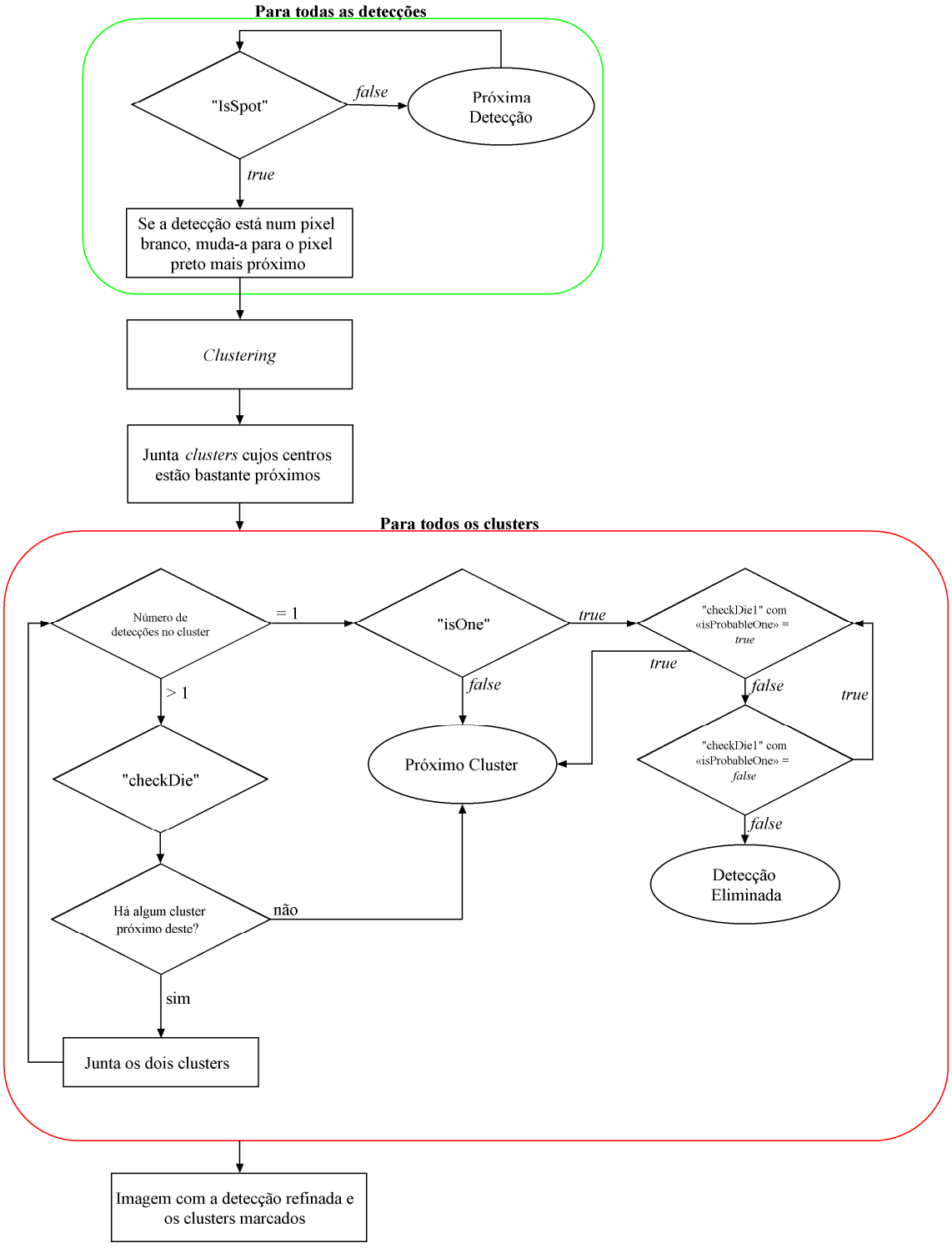


Imagem 10: Fluxograma do refinamento da contagem.

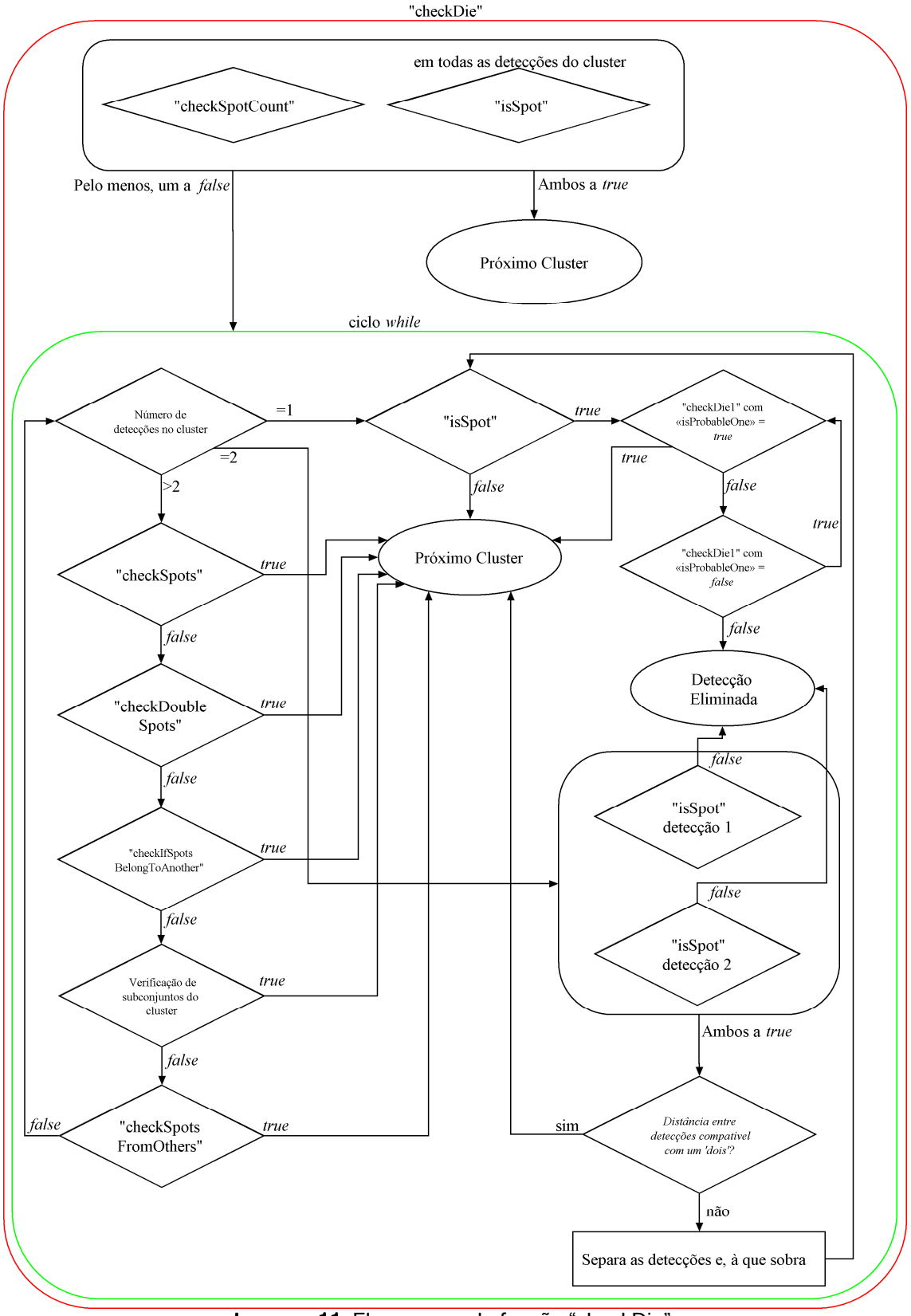


Imagem 11: Fluxograma da função “checkDie”.

2.3 - Estatística

Considerando que cada lançamento de dados é independente dos outros, a distribuição obtida a partir da contagem do resultado dos dados em cada lançamento é uma distribuição gaussiana, na medida em que se tem um valor médio esperado (para 14 dados, o valor mais provável é 49).
Um pouco de História: esta distribuição foi estudada, pela primeira vez, por DeMoivre (1667-1754) que estava curioso acerca do seu uso para prever as probabilidades em jogos de azar, nomeadamente de dados, o que é o caso desta experiência.
Esta distribuição tem a seguinte representação:

$$p(x) = y_0 + A\,e^{-\frac{(x-\mu)^2}{\sigma^2}} \qquad \text{(eq. 2)}$$

Em que:
- y_0 é o *offset* relativamente ao zero (espera-se que este valor seja muito próximo de zero);
- A é a amplitude da curva;
- μ é o valor médio da curva (espera-se que seja próximo de 49);
- σ é o valor do desvio padrão.

O ajuste às contagens obtidas é feito recorrendo ao método de Levenberg-Marquardt, pois este é um método simples (bastante semelhante ao método dos mínimos quadrados) e converge realtivamente depressa quando comparado com o dos mínimos quadrados.
Como valores iniciais, usam-se:
- para y_0, o zero;
- para A, escolhe-se a contagem que já foi efectuada mais vezes e usa-se esse número de vezes;
- para μ, usa-se 49, pois, para dados de seis faces, o valor médio desta distribuição é obtido pela fórmula: $\mu = \frac{6N + N}{2}$, em que N é o número de dados.
- para σ, usa-se $\mu/2$, apesar de haver uma forma de obter o valor esperado correcto (usando $\sigma = \sqrt{\frac{\sum_{i=1}^{K}(y_i - \mu)^2}{K}}$, em que K é o número de valores de y presentes, neste caso é 71)

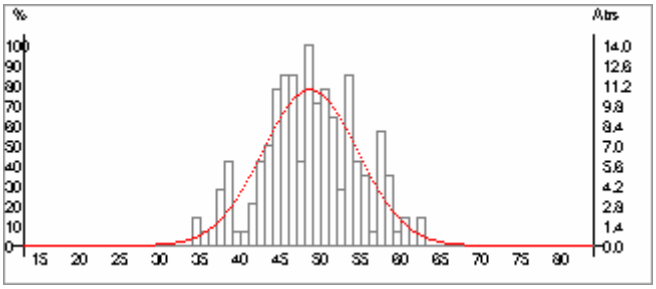


Imagem 12: Gráfico de barras de 162 contagens efectuadas pelo detector, com o ajuste da gaussiana sobreposto – valor médio: 49.17; desvio padrão: 5.91; *offset*: 0.01; Amplitude: 10.87; χ^2: 174.3.

3 – Experiência

3.1 - Aparato Experimental

A imagem a ser tratada é obtida usando uma webcam Logitech® QuickCam® Zoom. Esta webcam é capaz de adquirir imagens com uma resolução de até 640×480 pixeis, sendo esta a resolução da imagem que se vai analisar.

Para baralhar os dados usa-se um altifalante, virado para cima. Imediatamente sobre o cone deste altifalante está uma plataforma onde se colocam os dados. Esta plataforma tem a cor preta, pois os dados utilizados são brancos e têm pintas pretas. Assim, obtém-se o máximo contraste entre o fundo e os dados, tornando o processamento da imagem mais simples. Esta plataforma tem uma "parede" cilíndrica para evitar que os dados saiam pela borda; esta parede também é de cor preta.

Sobre tudo isto coloca-se a webcam, a uma distância tal que um dado que esteja em qualquer ponto da periferia da plataforma esteja incluído no campo de visão dela.

Sobre a webcam está um candeeiro de formato parabolóide. A lâmpada deste candeeiro fica posicionada mesmo atrás da webcam, com um reflector orientado para cima, o que evita que a imagem se torne excessivamente brilhante em alguns pontos. O formato parabolóide assegura a uniformidade da iluminação, mesmo apesar da webcam tapar uma parte do foco.

3.2 - Software

O *software* é composto por duas partes principais:

Server: Está a correr na máquina que comunica directamente com a experiência.

Client: Corre na máquina do utilizador. Comunica com o Server.

3.2.1 - *Server*

No *Server* está a correr o programa que interage com o *hardware* e envia/recebe as informações necessárias para/de os *Clients*.
Aqui é gerado o vídeo que pode ser visualizado por qualquer pessoa e que o Client vai usar para adquirir a imagem a analisar. Este video é uma sequência de imagens em jpeg e não um video real. Esta sequência é adquirida com um "frameRate" de cerca de 10 fps, simulando, assim, um video. Esta foi a melhor solução encontrada para enviar a informação do video através da plataforma do eLab, dado que esta plataforma foi desenhada para passar dados, em pouca quantidade e não para fazer streaming de video. Nas outras experiências que usam video, este é codificado usando o Windows Media Encoder e a imagem obtida delas não é usada nas experiências em si. Isto não é o caso nesta experiência em que é necessário trabalhar sobre a imagem obtida. Como não é possível adquirir uma imagem do Windows Media Encoder,

sem abrir o stream e não é possível abrir o stream em java, não é possível usar a imagem do encoder. Assim, não se usa o encoder, sendo o video fornecido directamente pela aplicação, do modo atrás indicado.
Após deixar de filmar, o programa adquire uma imagem para análise local. Apenas depois de proceder a esta análise é que esta imagem é enviada para o *client*, tal como o resultado da contagem das pintas. Caso seja a primeira imagem a ser enviada, também são enviados os dados da estatística acumulada e os valores de configuração do detector de pintas.
Daqui, passa à próxima amostra, caso tenha sido requisitada pelo utilizador.

3.2.2 - *Client*

O *Client* é o programa que corre no computador do utilizador.
O *Client* pode configurar alguns parâmetros da experiência.
Esses parâmetros são:
- A duração da onda sonora que baralha os dados => com um mínimo de 1.5 segundos e um máximo de 10 segundos;
- As frequências inicial e final da onda sonora => com um mínimo de 20 e um máximo de 150Hz. Estes valores foram determinados para que os dados vibrem sempre, por muito pouco que seja. Entre os 30 e os 60 Hz, a vibração é máxima;
- O número de amostras a tirar => com um mínimo de 1 e um máximo de 20. Este máximo é relativamente baixo, mas é para garantir que os utilizadores que estão à espera para tomar controlo da experiência não desesperem;
- Escolher ver ou não o video dos dados a serem baralhados => este é simplesmente um parâmetro sim/não e, por defeito, está desligado, para poupar largura de banda.
O *Client* tem uma janela de configuração (o customizer) , para a configuração destes parâmetros.

Na janela principal ("Image Analysis") aparece a imagem antes do processamento e, carregando nos vários botões disponíveis, a imagem é substituída pela correspondente ao botão carregado. Os botões são os seguintes:
- Original => Mostra a imagem original.
- B&W => Mostra a imagem em format binário (preto e branco), após o processaento necessário.
- Edges => Mostra a imagem dos contornos, após o processamento necessário.
- Hough => Mostra a imagem obtida pela aplicação da transformada de Hough, após o processamento necessário.
- Count => Mostra a imagem original, com as pintas marcadas. Também dá a indicação de quantas pintas foram encontradas, em "Dots Found", e de quantos dados foram detectados, em "Dice Found", após todo o processamento da imagem.
- Next Sample => Mostra a imagem seguinte, caso esteja disponível.

Todos os processamentos necessários para chegar às imagens visualizadas são feitos localmente, no computador do utilizador, logo o tempo de processamento depende deste computador. Deste modo, o utilizador tem percepção do peso computacional da análise de imagem.

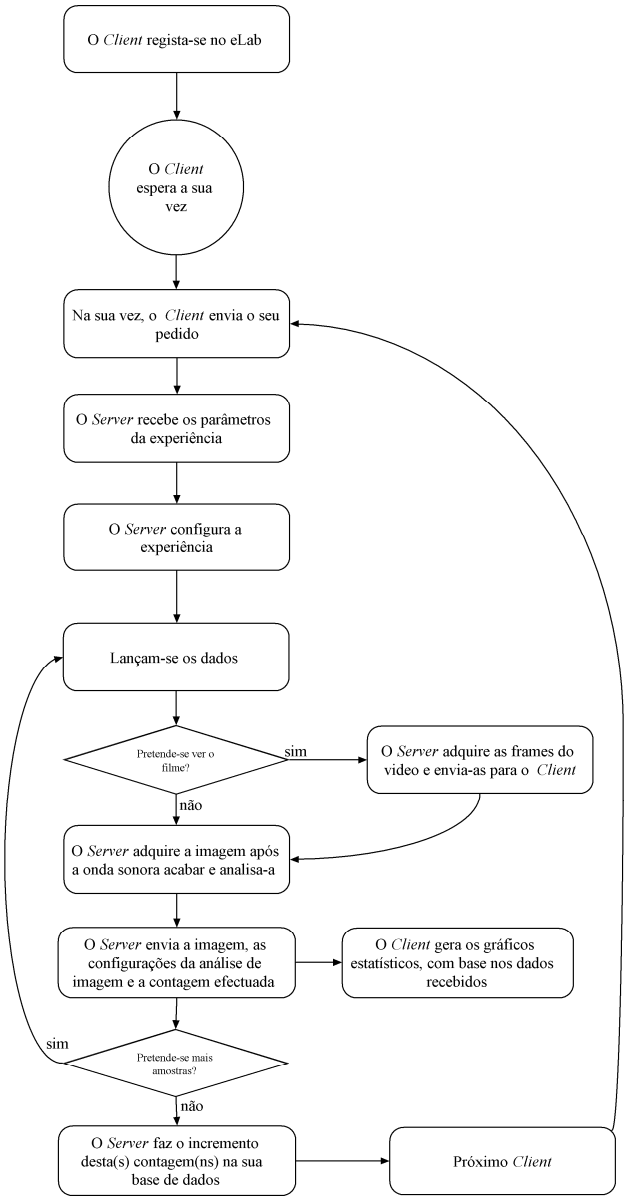
Na janela "Session Statistics", o utilizador poderá ver um gráfico de barras com os dados adquiridos durante a experiência corrente. Para além do gráfico de barras, também é desenhado o gráfico da função gaussiana, ajustada aos dados disponíveis.

Na janela "Accumulated Statistics", o utilizador poderá ver um gráfico de barras com os dados adquiridos até à experiência corrente e durante esta. Para além do gráfico de barras, também é desenhado o gráfico da função gaussiana, ajustada aos dados disponíveis. Como, neste gráfico, se tem mais dados para ajustar, o resultado do ajuste estará mais de acordo com a distribuição de probabilidades característica desta experiência.

Na janela "Movie", é visualizado o video. Este video é uma sequência de imagens em formato jpeg, que são adquiridas, no servidor, a cerca de 10 frames por segundo. Assim, nesta janela, as imagens são mostradas ciclicamente, a uma velocidade de cerca de 10 frames por segundo, também. Mesmo que ainda não tenham chegado todas as imagens, correspondentes ao video completo, as imagens que já estão disponíveis são mostradas.
Nesta Janela, há dois botões:
- Play/Pause => Quando o video está a andar, o botão mostra as duas barras típicas que indicam "pause"; se for carregado, o video pausa na imagem presente. Enquanto pausado, o botão mostra o triângulo típico que indica "play"; se for carregado, o video anda.
- Rewind => O botão mostra uma seta circular que, quando pressionado, reinicia o video.

Na janela "User Image Analysis", sem interagir com o Server, o Client pode correr o algoritmo de detecção de pintas em dados numa imagem que o utilizador tenha guardada no seu disco ou até numa imagem adquirida durante a experiência.
Este é um modo de operação do programa em que o utilizador pode entrar em contacto com os parâmetros da análise de imagem. Isto é porque esses parâmetros estão calibrados para as condições de iluminação presentes no eLab e que, muito provavelmente, não serão as mesmas que o utilizador terá nas suas imagens. Assim, o utilizador poderá regular o *threshold* de diferenciação entre claro e escuro, o raio esperado para as circunferências, os três *thresholds* de detecção dos centros das circunferências e o *threshold* de detecção de circunferências por convolução.
Como o Client não interage com o Server, todas as informações extraídas das imagens, neste modo, são apenas para consumo do utilizador e não contam para a estatística acumulada.

3.2.3 - Sequência do Programa Principal



- O *Client* regista-se no eLab.
- O *Client* espera pela sua vez. Enquanto espera, pode acompanhar os resultados das experiências que estiverem a correr.
- Na sua vez, o *Server* espera 10 segundos para que o utilizador envie o seu pedido. Caso não o faça nesse tempo, passa ao próximo utilizador.
- Quando é feito o envio do pedido, são enviados ao *Server* os parâmetros que o utilizador pretende para a experiência.
- O *Server* inicia a experiência e, depois de enviar a primeira imagem, envia também os dados acumulados de experiências passadas e os parâmetros de análise de imagem.

Uma amostragem é caracterizada por:

Lançamento dos dados;
Aquisição da imagem;
Análise da imagem no Server;
Envio da imagem e das informações obtidas pela análise para o *Client*.

- Por cada amostra, o *Client* gera um gráfico estatístico com as amostras recolhidas na presente experiência e, com a estatística acumulada, recebida do *Server*, gera um gráfico estatístico de todas as experiências já efectuadas.
- O *Server* faz o incremento das contagens obtidas nesta experiência na sua base de dados.
- O *Server* passa ao próximo ulitiazador e recomeça-se o processo.

4 – Conclusões

- A análise de imagem é uma boa ferramenta para adquirir diferentes tipos de dados, duma só vez.
  - Neste caso, isto acontece ao se determinar a quantidade de pintas presentes na imagem e a quantidade de dados.
  - Noutros casos, pode-se fazer outros tipos de análise sobre a imagem adquirida e obter muitos tipos diferentes de dados.

- A contagem das pintas de N dados segue uma distribuição poissoniana. No domínio contínuo, esta distribuição é representada pela gaussiana.

- O valor médio desta contagem, para N dados de 6 faces é de:

$$\overline{\mu} = \frac{6N + N}{2}$$

  - Para os 14 dados, μ=49, o que é verificado.

- O eLab é uma plataforma em crescimento, mas que já permite o controlo remoto de vários tipos de experiências, de carácter didático.
  Com o tempo, pode ser que esta plataforma possa ser utilizada em experiências mais avançadas, para serem usadas apenas por utilizadores autorizados.

- Esta experiência está disponível em:
  http://elab.ist.utl.pt/rec/eLab/Aleatorio/eLabAleatorioClient.jnlp

Bibliografia:

- http://www.symynet.com/educational_software/teaching_resources/Statistics/normal_distribution/intro.htm

- A. Mehta and J. M. Luck, *Novel Temporal Behavior of a Nonlinear Dynamical System: The Completely Inelastic Bouncing Ball*, Phys. Rev. Let. **65**, Nr.4, 393 (1990)