FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Inspeção Visual Automática de Peças Fundidas

Carlos Marcelo Silva Torres

Mestrado Integrado em Engenharia Mecânica

Orientador: Prof. João Manuel R.S. Tavares, FEUP

Coorientador: Eng. Valter Costa, INEGI

11 de Outubro de 2019

Inspeção Visual Automática de Peças Fundidas

Carlos Marcelo Silva Torres

Mestrado Integrado em Engenharia Mecânica

11 de Outubro de 2019

Resumo

Com o desenvolvimento da indústria, surge a necessidade de aumentar a eficiência de todas as etapas envolvidas no processo de produção de uma peça, em particular através da automatização das mesmas. A área da inspeção de qualidade é mais um exemplo onde a automação permite um melhor controlo dos critérios necessários ao bom desempenho do produto e/ou acordados entre fornecedor e cliente.

Nesta dissertação, são desenvolvidos algoritmos para detecção de defeitos superficiais em puxadores resultantes da fundição do alumínio, em particular defeitos nas arestas. No desenvolvimento deste projeto, são utilizadas técnicas de (i) pré-processamento da imagem para remover o ruído e localizar o objecto, (ii) extração de *features* para determinar as arestas e vértices da peça, (iii) segmentação para dividir o objecto em regiões de interesse, (iv) processamento da imagem para calcular um conjunto de métricas associadas a cada região de interesse e (v) análise estatística dos resultados.

No final, serão discutidos os resultados obtidos, avaliando a relevância de cada métrica como método de classificação da peça, expondo os problemas que surgem e o seu impacto no desempenho do algoritmo.

Abstract

With the development of the industry, there is a need to increase the efficiency of all steps involved in the production process of a part, in particular by automating them. The area of quality inspection is another example where automation allows a better control of the criteria necessary for good product performance and/or agreed between supplier and customer.

In this dissertation, algorithms were developed to detect surface defects in knobs resulting from aluminum casting, in particular edge defects. In the development of this project, we use (i) image preprocessing techniques to remove noise and locate the object, (ii) feature extraction to determine the edges and vertices of the part, (iii) segmentation to divide the object into regions of interest, (iv) image processing to calculate a set of metrics associated with each region of interest, and (v) statistical analysis of the results.

At the end, the results will be discussed, evaluating the relevance of each metric as a method of classification of the part, exposing the problems that arise and their impact on the performance of the algorithm.

Agradecimentos

Em primeiro lugar, quero agradecer ao Professor Dr. João Manuel Tavares por todo o apoio fornecido durante a elaboração desta dissertação, pela sua disponibilidade e orientação.

Ao Engenheiro Valter Costa pela supervisão, disponibilidade e esclarecimento de dúvidas que foram surgindo ao longo do trabalho.

Aos meus pais pela dedicação e compreensão prestados durante os meses em que foi desenvolvido este trabalho.

Carlos Torres

Conteúdo

1 Introdução **1**
- 1.1 Enquadramento do projeto . . . 1
- 1.2 Objectivos e Metodologia . . . 1
- 1.3 Estrutura da Dissertação . . . 2

2 Revisão Bibliográfica **3**
- 2.1 Inspeção Visual e Visão Artificial . . . 3
- 2.2 Imagem . . . 3
  - 2.2.1 Ruído . . . 5
- 2.3 Princípios básicos de Processamento de Imagem . . . 5
  - 2.3.1 Operações aritméticas . . . 5
  - 2.3.2 Thresholding . . . 5
  - 2.3.3 Operações lógicas . . . 6
  - 2.3.4 Detecção de contornos . . . 7

3 Desenvolvimento do algoritmo para detecção de defeitos **9**
- 3.1 Material de trabalho . . . 9
- 3.2 Remoção de ruído . . . 12
  - 3.2.1 Particularidade da configuração 9 . . . 16
- 3.3 Regiões de Interesse . . . 16
  - 3.3.1 Vértices . . . 16
  - 3.3.2 Arestas . . . 17
- 3.4 Métricas . . . 18
  - 3.4.1 Vértices . . . 19
  - 3.4.2 Arestas . . . 22
  - 3.4.3 Operador de Harris . . . 23
- 3.5 Outras funcionalidades . . . 24
- 3.6 Resumo . . . 24

4 Análise dos resultados obtidos **27**
- 4.1 Método utilizado . . . 27
- 4.2 Resultados obtidos . . . 28
  - 4.2.1 Interpretação dos parâmetros *P&R* . . . 29
  - 4.2.2 Comentário dos resultados . . . 30
  - 4.2.3 Limitações . . . 36
  - 4.2.4 Tempos de processamento . . . 36
- 4.3 Resumo . . . 38

5 Conclusões e Trabalho Futuro **39**
5.1 Satisfação dos Objetivos 39
5.2 Trabalho Futuro 40

Referências **41**

A Código Matlab **43**

B Resultados da peça 0 configuração 3 **87**

C Resultados da peça 2 configuração 3 **89**

D Resultados da peça 4 configuração 3 **91**

Lista de Figuras

2.1 Diferença entre imagem com elevada resolução espacial (esquerda) e imagem com baixa resolução espacial (direita) in (Solomon e Breckon, 2011). 4
2.2 Diferença entre imagem com 8 *bits* de quantização (esquerda) e 4 *bits* (direita) in (Solomon e Breckon, 2011). 4
2.3 Exemplo de soma e subtracção de uma constante a todos os pixeis de uma imagem. 5
2.4 Imagem original em tons de cinzento e respetiva binarização com *threshold* obtido pelo método de Otsu. 6
2.5 Exemplos de operações lógicas entre imagens. 6
2.6 Rebordos obtidos pelo método de a) *Roberts*, b) *Prewitt* e c) *Sobel*. 7
2.7 Rebordos obtidos pelo método de a) *LoG* e b) *Canny*. 8

3.1 Peças defeituosas fornecidas para a estudo de visão artificial. 9
3.2 Exemplos de imagens nas configurações relevantes: a) peça 4 configuração 3 (12ms), b) peça 4 configuração 4 (12ms), c) peça 4 configuração 5 (300ms), d) peça 2 configuração 9 (120ms). 11
3.3 Imagem original da peça 4 na configuração 3 (a) e respectivo resultado da equalização de histograma aplicado (b). 12
3.4 Contornos resultantes do método de *canny*. 13
3.5 Contornos obtidos pelo método de *Canny* dilatados. 14
3.6 Imagem com preenchimento de contornos fechados. 14
3.7 Montagem de imagens, realçando a verde os pixeis erodidos. 14
3.8 Imagem resultante após filtragem de áreas e perímetros. 14
3.9 Diagrama síntese da função *segmentation.m*. 15
3.10 Defeito presente na peça 4 configuração 9 (160ms). 16
3.11 Exemplo do posicionamento das regiões dos vértices através do algoritmo *RANSAC* aplicado aos pontos do contorno. 17
3.12 Exemplo da intersecção entre imagem binária e as rectas jericais na zona superior da peça, em que seria selecionado o lado esquerdo. 17
3.13 Resultado após seleção de regiões relevantes a analisar. 18
3.14 Exemplo de duas regiões de vértice, a da esquerda será considerada por estar do lado iluminado enquanto que a da direita será descartada. 19
3.15 Estimação de círculo utilizando o método de Pratt. 20
3.16 Exemplo representativo da estimação de parábolas no vértice. 20
3.17 Normais das orientações obtidas pelo algoritmo *skeletonOrientations.m*. 22
3.18 Recta estimada pelo *RANSAC* aplicado aos pontos de uma aresta com defeito. . . 22
3.19 Normais das orientações obtidas pelo algoritmo *skeletonOrientations.m* numa peça defeituosa. 23
3.20 Diagrama geral do algoritmo. 25

4.1 Distribuição da variável *circle_radius_relevant* de peça 4 configuração 3. 28
4.2 Representações gráficas das métricas relacionadas com o aresta da peça 4 configuração 3 e respectiva gama de aceitação selecionada. 31
4.3 Representações gráficas das métricas relacionadas com o vértice da peça 4 configuração 3 e respectiva gama de aceitação selecionada. 32
4.4 Comparação entre diferentes tempos de exposição luminosa no contorno da peça 0 com a configuração luminosa 3, em que a zona magenta representa a diferença entre imagens com os tempos especificados. 34
4.5 Representações gráficas das métricas relacionadas com o vértice da peça 4 configuração 9 e respectiva gama de aceitação selecionada. 35
4.6 Tempos de processamento da peça 4 configuração 3 (azul) e configuração 9 (amarelo). 37

Lista de Tabelas

3.1 Tabela de Configurações (Martins, 2018). 10
3.2 Tabela de Propriedades. 12
3.3 Tabela de regiões de interesse 18
3.4 Tabela resumo das métricas. 24

4.1 Gamas de aceitação estimadas com base nos resultados. 29
4.2 Tabela de precisão e revocação da peça 4 configuração 3. 33
4.3 Tabela de precisão e revocação da peça 0 configuração 3. 33
4.4 Tabela de precisão e revocação da peça 4 configuração 9. 33
4.5 Especificações do computador utilizado no desenvolvimento do algoritmo. 37
4.6 Tempos de processamento. 37

Abreviaturas e Símbolos

ACC	Accuracy
BACC	Balanced Accuracy
CPU	Central Processing Unit
FEUP	Faculdade de Engenharia da Universidade do Porto
FN	False Negative
FP	False Positive
LED	Light-Emitting Diode
MSE	Mean-Squared Error
OS	Operating System
PPV	Positive Predictive Value
P&R	Precisão e Revocação
RAM	Random Access Memory
RANSAC	Random Sample Consensus
REC	Revocação
STA	Sociedade Transformadora de Alumínios
SVM	Support-Vector Machine
TN	True Negative
TNR	True Negative Rate
TP	True Positive
UTAF	Unidade de Tecnologias Avançadas de Fabrico

Capítulo 1

Introdução

1.1 Enquadramento do projeto

Este projeto originou do interesse da Sociedade Transformadora de Alumínios (STA) em abordar o tema da visão artificial para melhorar a qualidade de produção das peças fundidas. A STA é uma empresa especializada no desenvolvimento e produção de sistemas para portas e janelas, sendo um dos seus principais processos de concessão a fundição do alumínio. Deste modo, a partir da sua parceria com o INEGI, em particular com o núcleo da UTAF, origina a ideia para desenvolver um sistema autónomo capaz de detectar os defeitos nas peças. Para este efeito foram fornecidas vários puxadores em alumínio fundido como objectos de teste.

Desde então, foi realizada uma primeira abordagem ao problema pelo meu colega Eng. João Martins, na qual resultou um estudo avançado sobre as condições relevantes no *setup* do projeto e a construção de um suporte mecânico automatizado para fotografar as peças. No final foi elaborada uma base de dados com as imagens das peças defeituosas em diferentes configurações *i.e.* orientação, esquema de iluminação e parâmetros de captura da câmara (tempo de exposição, ganho, *gamma* e valor da abertura da íris).

Tendo como ponto de partida esta base de dados, procedeu-se ao desenvolvimento de um algoritmo capaz de processar imagem e extrair métricas que serão estudadas para classificar a peça em análise com um bom nível de precisão. Com este efeito foi escolhido o *software* MATLAB como plataforma de desenvolvimento devido à utilidade da *toolbox* de Visão Computacional, abundante documentação (MATLAB, 2019) e experiência com o programa.

1.2 Objectivos e Metodologia

Este trabalho tem como principais objectivos o estudo de algoritmos computacionais de visão artificial usados em ambiente industrial e a sua implementação em casos reais.

Assim, após um estudo sobre o estado da arte, são selecionadas as configurações a serem estudadas nesta dissertação. A base de dados dispõe de dez configurações de iluminação diferentes

aplicadas a cinco peças com diferentes tipos de defeitos. Dentro de cada configuração, dispomos de um conjunto de imagens no qual variam parâmetros de captura como tempo de exposição à luz, gama, ganho, e abertura da iris da câmara. Destas, o foco foi em analisar algumas das configurações aplicadas à face frontal da peça, uma vez que cada uma poderá potencialmente exigir uma estratégia de processamento diferente ou pelo menos algumas permutações face a outras estratégias implementadas.

De seguida, são utilizados métodos de pré-processamento de imagem com o objectivo de remover ruido de fundo causado por pequenos reflexos da estrutura de ensaio e da garra que posiciona a peça. É também feita a localização do objeto na imagem aplicando técnicas de detecção de contornos e estimação de retas que permitem obter a localização aproximada dos cantos e arestas. Com estas informações a peça é dividida em regiões de interesse a serem analisadas separadamente, uma vez que dispomos de configurações com incidência lateral da luz e nesse caso pretende-se analisar apenas as arestas bem iluminadas em cada imagem. Além disso uma região que constitui um canto tem uma abordagem diferente de uma aresta. Em cada região de interesse serão extraídas métricas e será estudada a sua relevância no processo de classificação das peças.

1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 4 capítulos. No Capítulo 2 é feita uma revisão bibliográfica, abordando aspectos básicos relacionados com o processamento de imagem. No Capítulo 3 é explicado todo o processo realizado na elaboração do algoritmo. No Capítulo 4 são analisados os resultados obtidos e discutidas as limitações encontradas. No Capítulo 5 é feita uma conclusão final e proposta de trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

2.1 Inspeção Visual e Visão Artificial

A inspeção visual é uma prática que tem vindo a ser cada vez mais aplicada nas cadeias de produção. Tipicamente o produtor tem um conjunto de critérios de qualidade mínimos a serem garantidos no fabrico dos seus produtos que são acordados com o cliente. Este controlo de qualidade é muitas vezes efetuado manualmente por pessoas treinadas e com auxílio de ferramentas medidoras ou de iluminação. A qualidade da inspeção depende assim da acuidade visual do operador, fadiga, factores pessoais e sentimentais do operador.

Com o desenvolvimento industrial e o aumento da competitividade dos mercados globais, a capacidade de manter critérios de qualidade elevados de forma eficaz e rentável torna-se essencial para o sucesso da empresa. Numa altura em que muitas das etapas de produção (concessão, acabamento, transporte, armazenamento, logística, etc.) são automatizadas, o método tradicional de realizar a inspeção do produto pode tornar-se um *bottleneck* (Iglesias et al., 2018).

Surge assim o interesse em incorporar tecnologias de Visão Artificial no processo de inspeção, aumentando desta forma a rapidez de inspeção, reprodutibilidade e rentabilidade. Estes sistemas são habitualmente implementados em conjunto com sistemas de classificação baseados em algoritmos de inteligência artificial, permitindo ao mesmo tempo um melhor registo dos produtos, manipulação e interpretação das bases de dados construídas (Iglesias et al., 2018).

2.2 Imagem

Uma imagem pode ser definida como uma função $f(x,y)$ em que x e y são coordenadas espaciais e a amplitude f representa a intensidade da imagem no ponto correspondente (Gonzalez e Woods, 2002). Esta representação assume tipicamente um carácter discreto que resulta do funcionamento das câmaras digitais. Estas possuem um sensor que captura luz e acumula carga em função da intensidade luminosa em cada secção do sensor. Cada uma destas secções corresponde a um pixel, e o seu valor é obtido medindo a carga acumulada. O processo de discretização das

coordenadas da imagem é referido como *quantização espacial*, enquanto que o processo de discretização da amplitude é a *quantização da intensidade*. Estes termos, por vezes, são mencionados na literatura de forma corrente como *amostragem* e *quantização* respectivamente.

Uma imagem digital $I(m,n)$ é assim composta por um plano bidimensional de pixeis (m,n) em que a resolução espacial é definida pelo número de pixeis sob a forma $(m \text{ x } n)$ e a resolução de *bits* corresponde aos *bits* necessários para armazenar o número de intensidades que é possível obter em cada pixel (p.e. uma resolução de 8 *bits* permite a imagem ter um máximo de 256 tons de cinzento).

Na Figura 2.1 podemos ver um exemplo que compara imagens com diferente resolução espacial e na Figura 2.2 com diferente resolução de quantização.

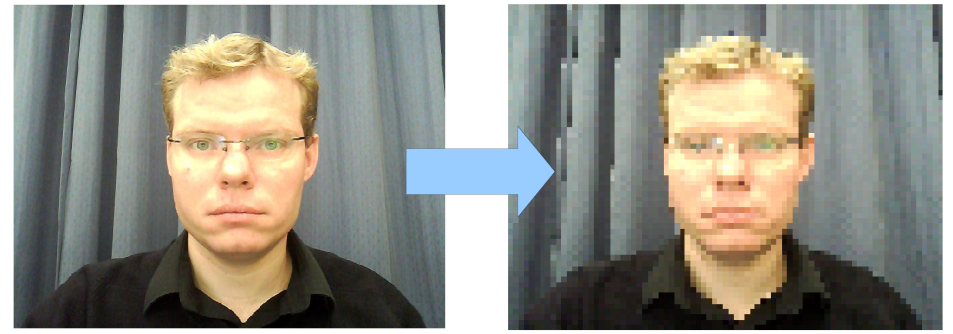


Figura 2.1: Diferença entre imagem com elevada resolução espacial (esquerda) e imagem com baixa resolução espacial (direita) in (Solomon e Breckon, 2011).

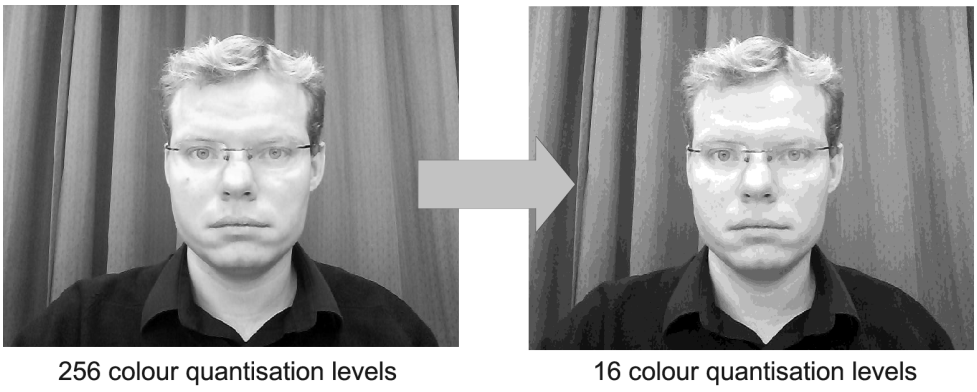


Figura 2.2: Diferença entre imagem com 8 *bits* de quantização (esquerda) e 4 *bits* (direita) in (Solomon e Breckon, 2011).

Embora uma imagem com elevada resolução permita armazenar um melhor nível de detalhe sobre o objecto em análise, isto nem sempre é favorável uma vez que o algoritmo de processamento terá de efetuar um maior número de operações. Deste modo, a resolução deverá ser apenas a suficiente para representar nitidamente os elementos de interesse.

2.2.1 Ruído

O ruído é o principal problema que surge no processamento de imagem. Por ruído entende-se a variação do sinal do seu valor verdadeiro em quantidades tipicamente pequenas e aleatórias (Solomon e Breckon, 2011). Assim sendo, a remoção de ruído é um dos principais componentes de qualquer sistema de processamento de imagem. Este pode ter diferentes origens, sendo as mais importantes (Solomon e Breckon, 2011):

- Ruído de captura — resulta de variações de luminosidade, temperatura do sensor, ruído electromagnético, pó, vibrações, saturações do sensor, etc;
- Ruído de amostragem — resulta das limitações de amostragem e de quantização.

2.3 Princípios básicos de Processamento de Imagem

Nesta secção serão descritas algumas das operações básicas de processamento de imagem que foram relevantes no projecto para o melhor enquadramento do leitor.

2.3.1 Operações aritméticas

Em ambiente de programação, uma imagem assume uma representação matricial. A forma mais simples de manipular uma imagem é realizando operações pixel a pixel. Desta forma podemos somar, subtrair, multiplicar ou dividir duas imagens ou uma imagem e um escalar.

A soma e a subtração de imagens são tipicamente usadas em operações em que se pretende manipular o brilho. Somando um valor constante a uma imagem vai aumentar a claridade e subtrair vai escurecer (Figura 2.3).

(a) I (b) I + constante (c) I - constante

Figura 2.3: Exemplo de soma e subtracção de uma constante a todos os pixeis de uma imagem.

A multiplicação e divisão funcionam de forma similar, ajustando o brilho proporcionalmente.

2.3.2 Thresholding

Thresholding é uma das técnicas mais usadas no processamento de imagem. Esta permite a transformação de uma imagem em tons de cinzento numa imagem binária (a preto e branco). O

critério de separação é um valor definido — *threshold* — que reflete a intensidade acima do qual os pixeis passarão a ter o valor 1 e abaixo serão 0. O *threshold* pode ser global, ou seja, o mesmo valor é usado para toda a imagem, ou adaptativo, em que é calculado um threshold individual para cada pixel com base nos pixeis adjacentes.

A Figura 2.4 mostra um exemplo de uma binarização obtida pelo método de Otsu (Otsu, 1979).

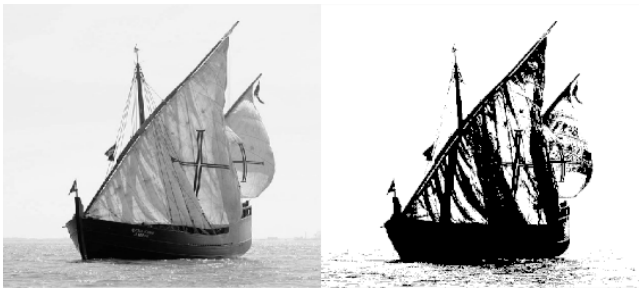
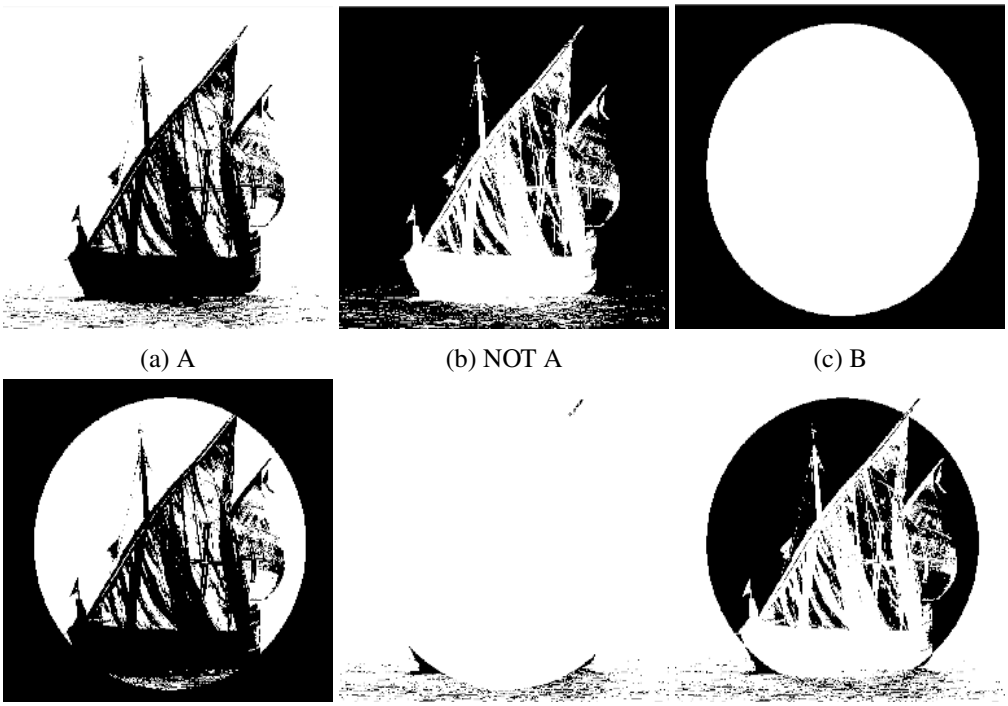


Figura 2.4: Imagem original em tons de cinzento e respetiva binarização com *threshold* obtido pelo método de Otsu.

2.3.3 Operações lógicas

É também possível realizar operações lógicas como *NOT*, *AND*, *OR* e *XOR* entre imagens. A Figura 2.5 contem exemplos de cada uma destas operações.



(a) A (b) NOT A (c) B

(d) A AND B (e) A OR B (f) A XOR B

Figura 2.5: Exemplos de operações lógicas entre imagens.

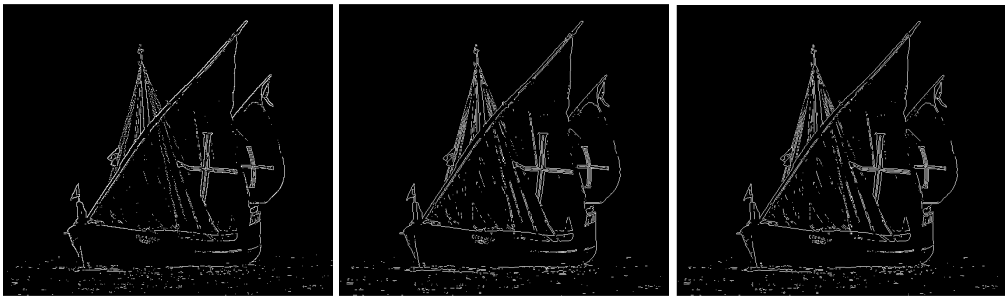
2.3.4 Detecção de contornos

Um contorno numa imagem é definido como uma descontinuidade ou um gradiente presente na mesma. Estas descontinuidades são mudanças bruscas na intensidade dos pixeis e caracterizam fronteiras de objectos presentes na imagem. Desta forma as técnicas de detecção de contorno operam à base de filtros derivativos, os quais podem ser divididos em filtros de primeira ordem e de segunda ordem (Solomon e Breckon, 2011).

Com o objectivo de aproximar uma medida da primeira ou segunda derivada da imagem, são utilizados *kernels* de convolução. Convolução é um processo que permite adicionar o valor de um pixel aos pixeis adjacentes com um peso que é atribuído pelo *kernel*, que frequentemente assume a forma de uma matriz 3×3 ou 5×5.

2.3.4.1 Operadores de primeira ordem

Os operadores de primeira ordem mais utilizados são *Sobel*, *Prewitt* e *Roberts* (Solomon e Breckon, 2011). Um exemplo que compara estes métodos pode ser visto na Figura 2.6.



(a) (b) (c)

Figura 2.6: Rebordos obtidos pelo método de a) *Roberts*, b) *Prewitt* e c) *Sobel*.

O operador de *Roberts* (Roberts, 1965) foi dos primeiros operadores a ser desenvolvido e permite uma detecção rápida e eficaz de alguns contornos da imagem. Devido ao operador 2×2, este método reage optimamente a contornos com orientação de ± 45 com o eixo horizontal.

Prewitt (Prewitt, 1970) e *Sobel* (Sobel, 1970) são operadores mais utilizados por possuírem melhor desempenho. A diferença entre estes reside no facto de o operador de *Sobel* implementar diferenciação numa direção e uma média Gaussiana na outra direção, o que suaviza a imagem e permite reduzir a interferência do ruído na detecção do contorno (Solomon e Breckon, 2011).

Outra propriedade interessante destes dois operadores, é o facto de ser possível expressar o *kernel* (matriz 3×3) através do produto matricial entre um vector linha (3×1) e um vector coluna (1×3). Isto permite processar a imagem através da aplicação do primeiro vector e ao resultado obtido aplicar o segundo vector, produzindo o mesmo resultado e reduzindo o número de operações aritméticas efetuadas (Solomon e Breckon, 2011).

2.3.4.2 Operadores de segunda ordem

O operador de segunda ordem mais utilizado é o *Laplaciano*. Este método procura as raízes na segunda derivada da imagem para determinar os contornos, obtendo assim uma melhor resposta nos pontos onde o gradiente local muda mais rapidamente (Maini e Aggarwal, 2009).

Uma das limitações deste método é a sensibilidade ao ruído. Deste modo este filtro é mais frequentemente aplicado em conjunto com um filtro de *Gauss* que suaviza a imagem, reduzindo a influência do ruído. Como a convolução é associativa, é possível combinar os dois *kernels* num só, originando o filtro *Laplacian of Gaussian (LoG)* (Solomon e Breckon, 2011).

Na Figura 2.7 pode ser vista uma comparação entre o método de *Sobel*, *LoG* e *Canny*.

2.3.4.3 Método de Canny

O método de *Canny* (Canny, 1986) é atualmente a forma mais utilizada para detectar os contornos dos objectos. Este funciona executando várias etapas de processamento diferentes, o que o torna um dos algoritmos mais complexos nesta área.

De forma resumida, este método procura (Solomon e Breckon, 2011):

- Efetuar a detecção minimizando a estimativa do erro;
- Produzir uma linha fina localizada no centro do contorno;
- Produzir apenas uma linha para cada contorno.

A Figura 2.7 contém um exemplo da aplicação deste método.

(a) (b)

Figura 2.7: Rebordos obtidos pelo método de a) *LoG* e b) *Canny*.

Capítulo 3

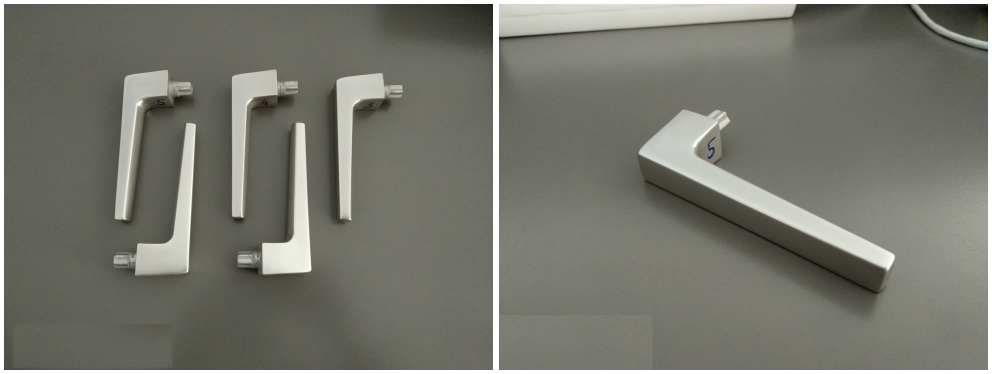
Desenvolvimento do algoritmo para detecção de defeitos

Neste capítulo são descritas todas as etapas do desenvolvimento do algoritmo.

Inicialmente será feita uma descrição do material disponível como ponto de partida, seguidamente será explicado o processo de selecção de configurações e só então será explicada a abordagem tomada no processamento de imagem. Esta abordagem encontra-se dividida em três etapas: remoção de ruído, detecção das regiões de interesse e cálculo de métricas para avaliação da integridade da peça.

3.1 Material de trabalho

Assim como foi referido em Secção 1.1, este projecto foi iniciado a partir de uma base de dados composta por 74 arquivos de fotografias tiradas a 5 peças defeituosas fornecidas pela STA (Figura 3.1).



(a) (b)

Figura 3.1: Peças defeituosas fornecidas para a estudo de visão artificial.

Estes arquivos encontram-se divididos por peça (0–4) e por configuração (1–10). Para este efeito, uma configuração representa um conjunto único de tipo de iluminação aplicada e orientação da peça. Na Tabela 3.1 estão descritas as configurações existentes.

Tabela 3.1: Tabela de Configurações (Martins, 2018).

Número	Iluminação	Orientação
1	Barra de LED Effilux	lado esquerdo (tangencial)
2	Barra de LED Effilux	lado direito (tangencial)
3	Barra de LED Effilux	frontal/angulo elevado (directa, $>45º$)
4	Barra de LED Effilux	angulo baixo ($<45º$)
5	Barra de LED Effilux	iluminação indirecta, posição 1
6	Barra de LED Effilux	iluminação indirecta, posição 2
7	Barra de LED Effilux	lado de trás, (backlight)
8	Domo LED vermelho	frontal
9	Anel de LED Neopixel	frontal
10	Domo rande LED Neopixel	frontal

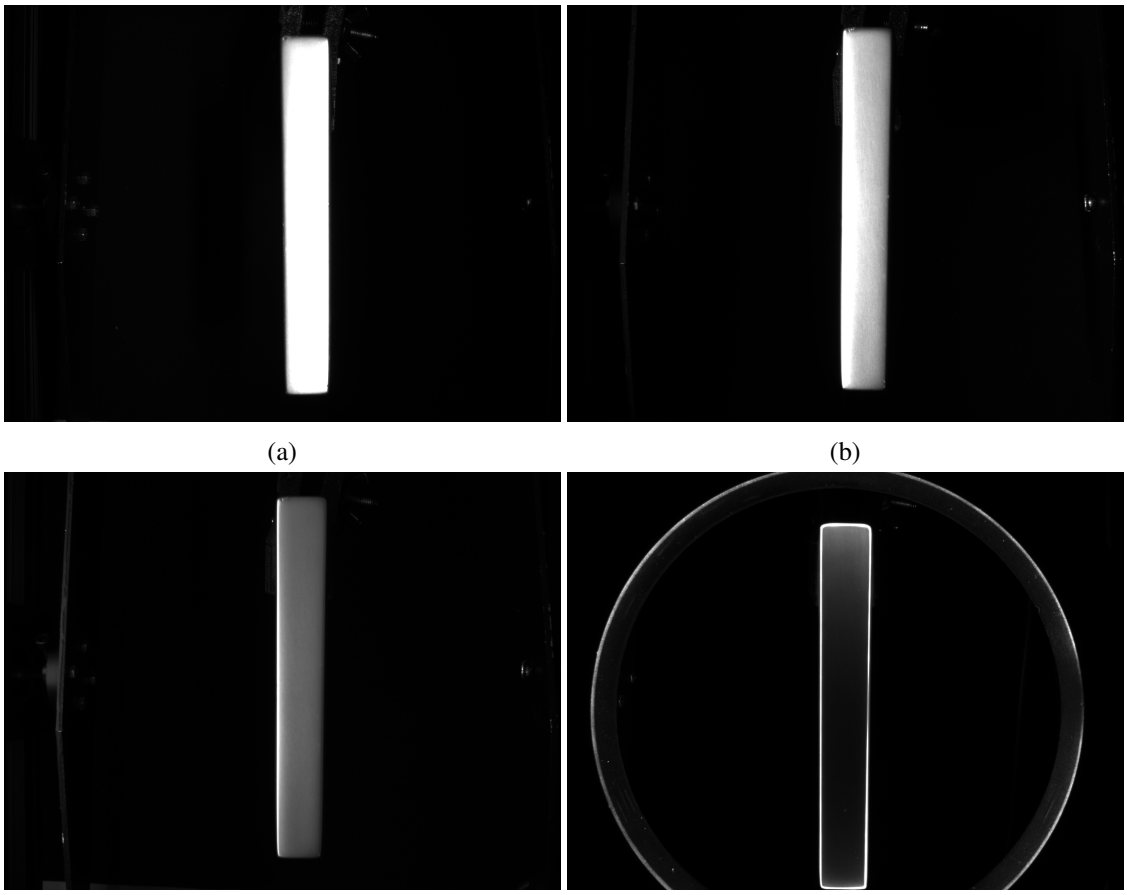
Após uma análise do material disponível foi necessário tomar duas decisões: face e tipo de defeitos a detectar. Assim, foram selecionadas as imagens correspondentes à face frontal da peça como objecto de estudo, uma vez que cada face e cada configuração requer um plano de processamento diferente.

Deste modo, foram apenas consideradas as configurações 3, 4, 5, 6, 9 e 10, e depois de rever novamente as imagens concluiu-se que:

- As config. 3 e 4 apresentam incidência da luz lateral, ficando bem visível a aresta iluminada, vértices adjacentes e defeitos interiores;
- A config. 5 apresenta incidência da luz lateral, ficando bem visível a aresta iluminada;
- A config. 6 apresenta incidência da luz lateral, ficando minimamente visíveis as arestas e bem visíveis os defeitos interiores;
- A config. 9 apresenta todo o contorno da peça bem visível;
- A config. 10 apresenta toda a face iluminada revelando defeitos interiores e má visibilidade do contorno.

Com estas informações optou-se por desenvolver um algoritmo focado na detecção de defeitos no contorno da peça, sendo as configurações mais importantes as 3, 4, 5 e 9. Na Figura 3.2 podemos ver um exemplo de cada uma.

Analisando a Figura 3.2 podemos notar rapidamente que a imagem d) se encontra cortada na aresta inferior. Isto trata-se de um descuido no alinhamento da captura e será discutido na Subsecção 4.2.3.



(a) (b)

(c) (d)

Figura 3.2: Exemplos de imagens nas configurações relevantes: a) peça 4 configuração 3 (12ms), b) peça 4 configuração 4 (12ms), c) peça 4 configuração 5 (300ms), d) peça 2 configuração 9 (120ms).

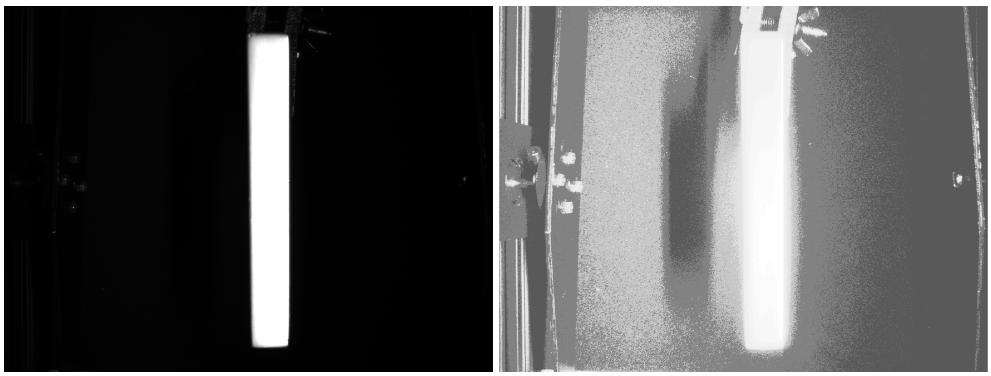
Cada um dos arquivos relevantes contém entre 8 a 24 imagens, variando parâmetros de captura como o tempo de exposição, gama, ganho e abertura da iris. A Tabela 3.2 sintetiza as propriedades das imagens que constituem a base de dados.

Tabela 3.2: Tabela de Propriedades.

Característica	Valor
Resolução	1542×2056
Bits por pixel	8 *bits*

3.2 Remoção de ruído

O primeiro passo para iniciar a construção do algoritmo é localizar a peça na imagem. Aplicando o método de *Canny* para detecção de contornos obtemos resultados que incluem elementos indesejáveis como reflexões na estrutura mecânica, vestígios do sistema de iluminação ou da garra que segura a peça. Estes elementos podem ser vistos aplicando uma equalização do histograma à imagem original, criando um melhor enquadramento do problema (Figura 3.3).



(a) (b)

Figura 3.3: Imagem original da peça 4 na configuração 3 (a) e respectivo resultado da equalização de histograma aplicado (b).

Como solução optou-se por utilizar um método sugerido pela documentação do MATLAB (Mathworks, 2019) e adaptar ao objecto deste projecto. Inicialmente efetua-se a detecção de contornos usando o método de *Canny*, no entanto multiplica-se os *thresholds* usados no processo por um factor (variável *fudge* em *main.m* disponível no Anexo A) que inicialmente toma o valor de 0.5. Isto permite extrair mais linhas da figura através da redução dos *thresholds* (Figura 3.4).

De seguida procedeu-se a uma dilatação da imagem binária obtida de 1 pixel em todas as direções, fechando potenciais aberturas de pequena dimensão (Figura 3.5).

Os contornos fechados presentes na imagem foram preenchidos utilizando a função *imfill* presente no *toolbox* de visão computacional como pode ser observado na Figura 3.6.

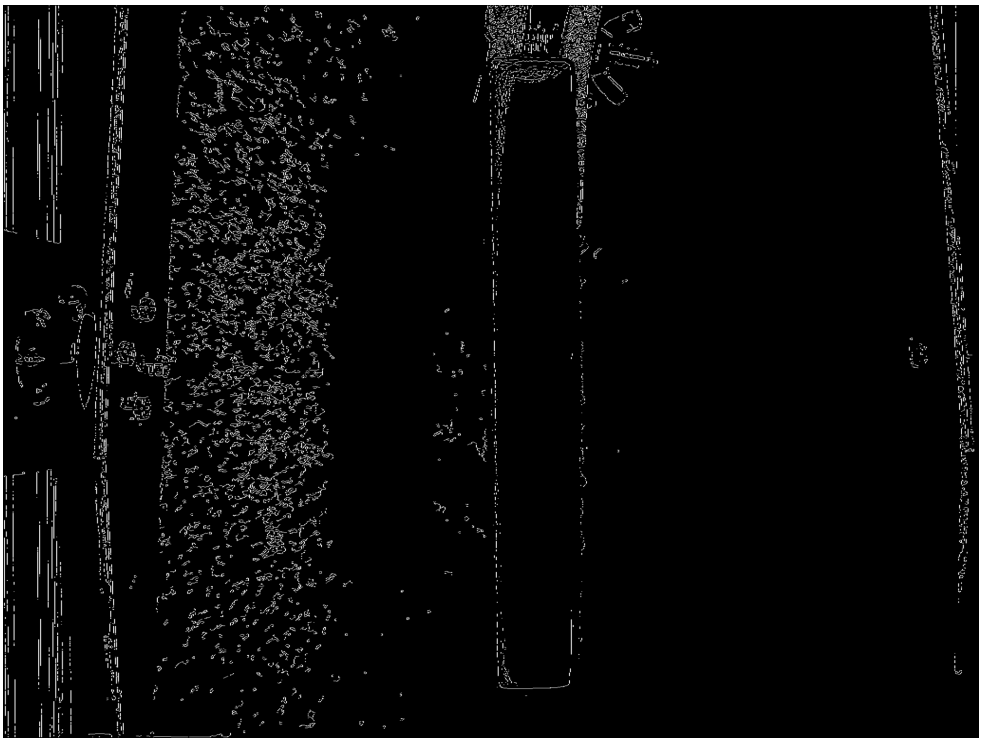


Figura 3.4: Contornos resultantes do método de *canny*.

Os elementos da figura são erodidos, retirando 1 pixel ao contorno de cada elemento, restaurando as dimensões originais do objecto (Figura 3.7).

Finalmente os elementos fechados resultantes dos processos anteriores são filtrados por áreas e perímetros. Sabendo que a área esperada da peça na imagem é de cerca de 23×10^4 pixeis, foi usado um valor mínimo de 12×10^4 (cerca de metade) e máximo de 26×10^4, removendo elementos com dimensão fora deste intervalo. Em configurações em que o suporte de iluminação é mais intrusivo (Figura 3.2 d.), é possível a detecção de múltiplos objectos. Para evitar este problema é também efetuada uma filtragem por perímetros com uma gama de aceitação de 2700 a 3400 pixeis, mantendo uma razão entre área e perímetro perto de 76. Isto parte do pressuposto que a distância da câmara à peça durante o processo de captura será relativamente constante. O resultado obtido deste processo pode ser observado na Figura 3.8.

Para concluir retira-se o contorno dos elementos da figura, que idealmente será apenas um — a peça.

Se por algum motivo não resultar nenhum elemento deste processo, o mesmo será repetido com um factor *fudge* mais baixo, permitindo extrair um maior número de linhas na fase de aplicação do *Canny*.

O diagrama da Figura 3.9 resume o processo descrito e o código MATLAB pode ser consultado no Anexo A sob a função *segmentation*.

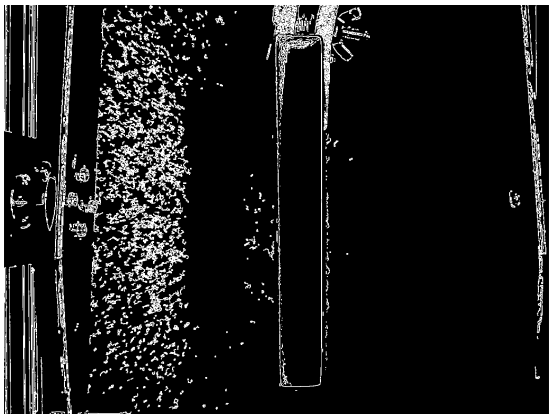


Figura 3.5: Contornos obtidos pelo método de *Canny* dilatados.

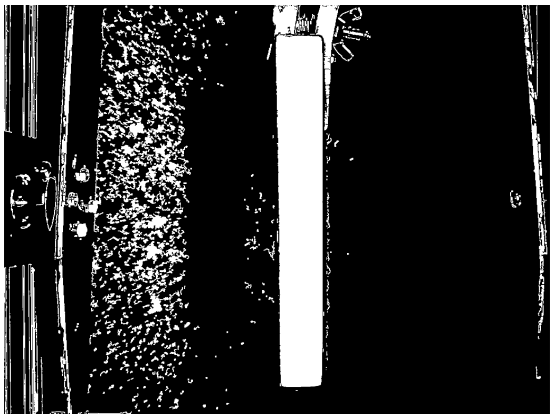


Figura 3.6: Imagem com preenchimento de contornos fechados.

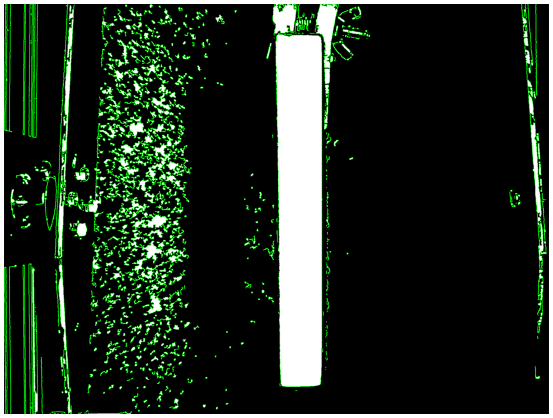


Figura 3.7: Montagem de imagens, realçando a verde os pixeis erodidos.

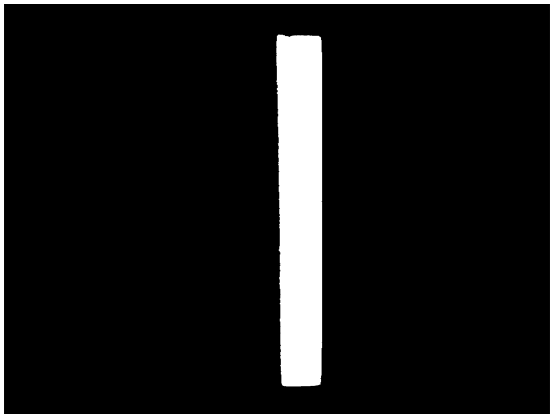


Figura 3.8: Imagem resultante após filtragem de áreas e perímetros.

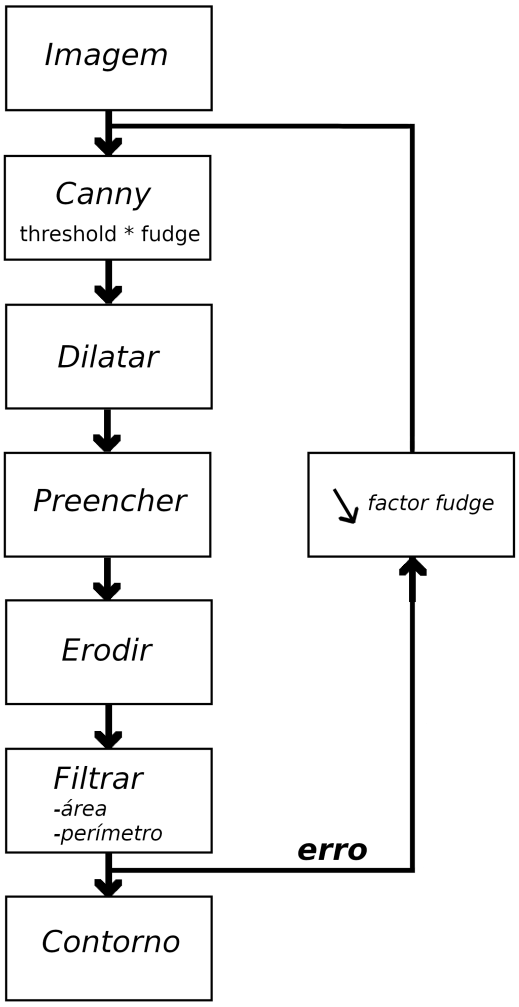


Figura 3.9: Diagrama síntese da função *segmentation.m*.

3.2.1 Particularidade da configuração 9

No caso da configuração 9, em que é utilizado o *dark field ring* como método de iluminação, o resultado apresenta apenas o contorno da peça iluminado. Utilizando a mesma abordagem, obtemos a linha exterior do contorno. No entanto, a linha interior também terá interesse analisar, uma vez que há defeitos que interferem com a continuidade de apenas uma das linhas como é exemplo o defeito da Figura 3.10.

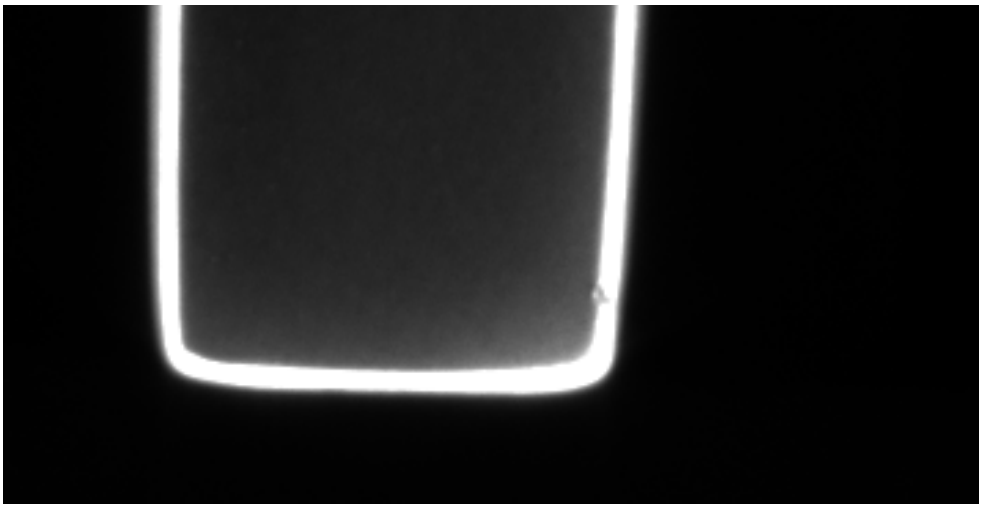


Figura 3.10: Defeito presente na peça 4 configuração 9 (160ms).

Deste modo, para obter a linha interior, utiliza-se uma permutação do mesmo algoritmo em que numa primeira fase é extraída a linha exterior. Esta será posteriormente subtraída ao resultado obtido pelo algoritmo de detecção de contornos *Canny* e o processo é repetido novamente obtendo a linha interior (função *segmentation9.m* em Anexo A).

3.3 Regiões de Interesse

3.3.1 Vértices

Estando o contorno da peça bem definido, utiliza-se o método *RANSAC* (Fischler e Bolles, 1981) para estimar as 4 rectas que constituem a geometria principal do objecto (função *line_fit.m* no Anexo A). Estas rectas são usadas para calcular a localização aproximada dos quatro vértices.

Por fim é desenhado um quadrado 50×50 pixeis e centro no vértice respectivo. Todos os pontos do contorno interiores ao quadrado passarão a pertencer a esse vértice para efeitos de calculo de métricas como será visto na secção Secção 3.4.

Na Figura 3.11 encontra-se um exemplo em que além das rectas obtidas pelo algoritmo *RANSAC*, estão representados os pontos utilizados na sua estimação (*inliers*) e as regiões de vértice localizadas na parte superior da peça.

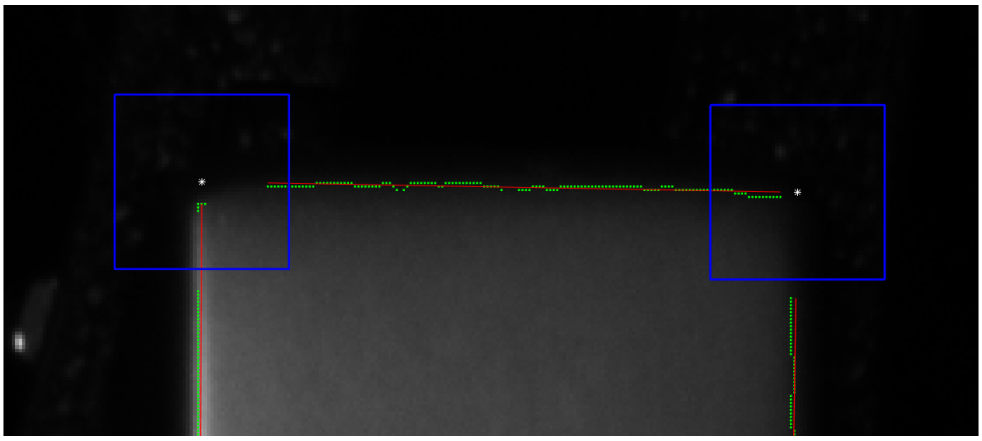


Figura 3.11: Exemplo do posicionamento das regiões dos vértices através do algoritmo *RANSAC* aplicado aos pontos do contorno.

3.3.2 Arestas

Como a maior parte das configurações possuem incidência lateral da luz, opta-se por apenas analisar a aresta iluminada em cada imagem (juntamente com os dois vértices adjacentes). Deste modo, para localizar o lado iluminado de forma automática, é efetuada uma intersecção entre as duas rectas verticais mencionadas na Subsecção 3.3.1 e uma binarização da imagem original com *threshold* obtido pelo método de *Otsu*. A recta que atravessar o maior número de pixeis da binarização será a recta de interesse. Um exemplo pode ser visualizado na Figura 3.12 e o código está presente em *line_fit.m* no Anexo A.

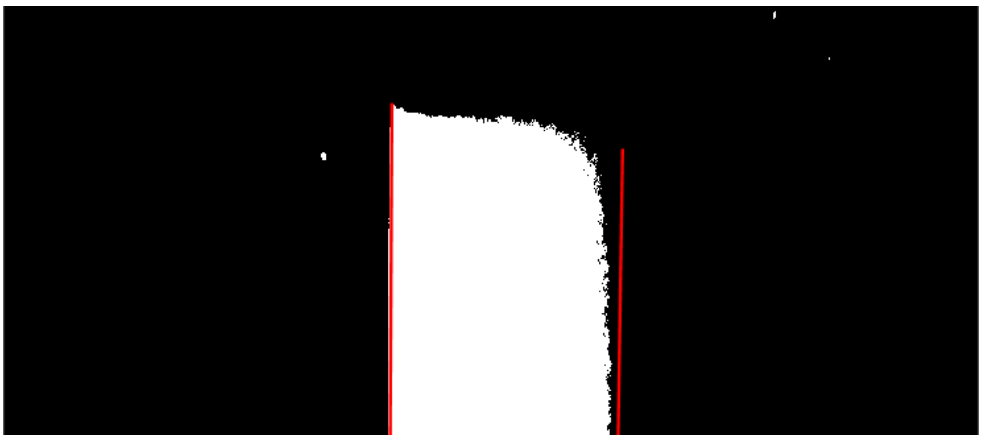


Figura 3.12: Exemplo da intersecção entre imagem binária e as rectas jericais na zona superior da peça, em que seria selecionado o lado esquerdo.

Com o modelo da recta "iluminada" são calculados os dois vértices mais próximos. Partindo das coordenadas dos vértices é desenhado um rectângulo que cobre toda a aresta de interesse sem intersectar o quadrado do vértice (zona azul claro da Figura 3.13). O código encontra-se em *rect.m* no Anexo A.

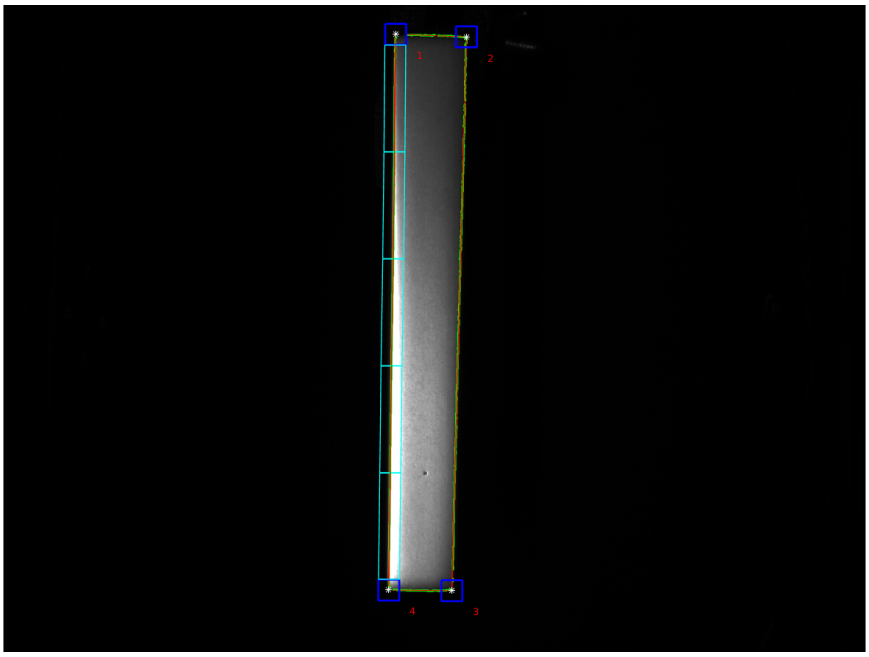


Figura 3.13: Resultado após seleção de regiões relevantes a analisar.

Os pontos interiores ao rectângulo são posteriormente ordenados segundo a orientação da recta e divididos em cinco grupos.

No caso de ser analisada a aresta superior como acontece no processamento das imagens da configuração 9, esta é apenas dividida em 3 regiões devido ao seu comprimento ser mais reduzido. Cada um destes grupos será analisado individualmente para detecção de defeitos como será explicado na Secção 3.4.

3.4 Métricas

Para poder interpretar a informação extraída da imagem foram calculados um conjunto de métricas que serão posteriormente estudadas quanto à sua influência no processo de classificação das peças. Estas, são determinadas por região e por geometria (vértice ou aresta).

Na Tabela 3.3 está representado o número regiões em cada configuração.

Tabela 3.3: Tabela de regiões de interesse

Configuração	Número regiões aresta	Número regiões vértice
config 3	5	2
config 4	5	2
config 6	5	2
config 9	16	4

3.4.1 Vértices

Cada região de vértice é constituída pelos pontos do contorno interiores ao quadrado desenhado na posição aproximada do vértice, como se encontra representado na Figura 3.14.

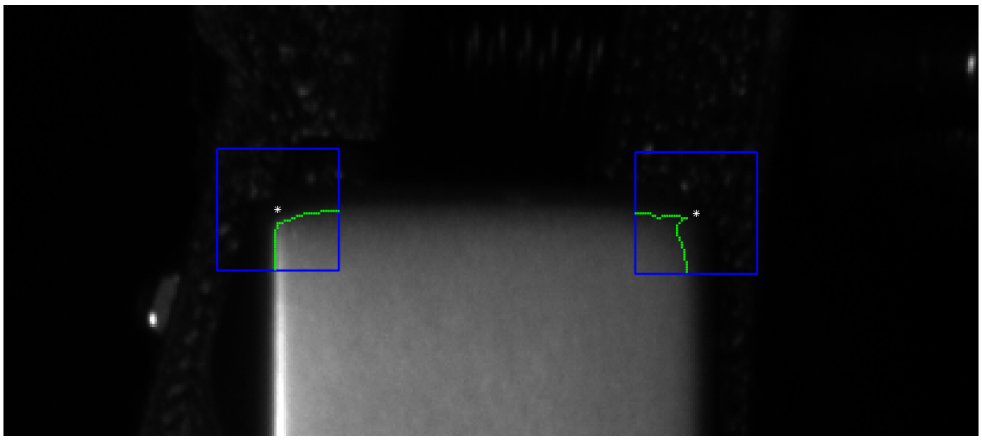


Figura 3.14: Exemplo de duas regiões de vértice, a da esquerda será considerada por estar do lado iluminado enquanto que a da direita será descartada.

3.4.1.1 Distâncias

Utilizando os pontos da região de interesse, é utilizado um algoritmo para estimar um circulo pelo método de Pratt, que é uma implementação robusta de estimação baseada nos mínimos quadrados (Pratt, 1987). Um exemplo de aplicação está representado na Figura 3.15.

Deste modo, para cada vértice obtemos o modelo do circulo (centro e raio), do qual é registado o raio na variável *circle_radius*. A variável *circle_radius* constitui os raios dos quatro vértices do objecto e a *circle_radius_relevant* contém apenas os raios dos vértices relevantes (iluminados). Esta nomenclatura segue a mesma filosofia para as restantes variáveis mencionadas neste documento.

Com o modelo do círculo são calculadas as distâncias de cada ponto ao circulo segundo a Equação 3.1.

$$distancia = |\sqrt{(x_1 - h)^2 + (y_1 - k)^2} - r| \tag{3.1}$$

Onde:

(x_1, y_1) = coordenadas do ponto
(h, k) = coordenadas do centro do círculo
r = raio do círculo

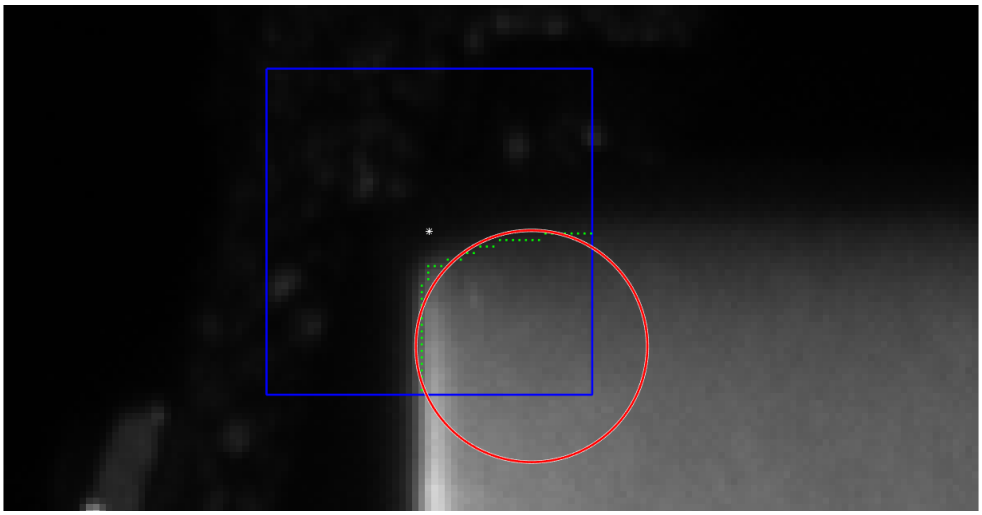


Figura 3.15: Estimação de círculo utilizando o método de Pratt.

Estas distancias formam um vector *circle_distout*, que é usado para calcular a média das distâncias (*circle_distout_mean*), o desvio padrão (*circle_distout_std*) e o número de pontos cuja distancia se encontra acima de um *threshold* de 3 pixeis (*circle_distout_outliers*).

Como a tangência do círculo aos pontos da região não é perfeita, optou-se por utilizar um segundo método de análise cujo desempenho será posteriormente comparado ao primeiro. Deste modo, é feita a estimação de duas parábolas pelo método *RANSAC*, que irão, idealmente, acomodar melhor a geometria da região.

De forma a evitar a sobreposição de ambas as parábolas na extremidade, é inicialmente feita a estimação de uma recta vertical pelo método *RANSAC* e isolados os pontos próximos da recta (*inliers*). Aos restantes pontos é feita a estimação de uma parábola que irá ficar localizada na zona da aresta superior como podemos ver no exemplo da Figura 3.16.

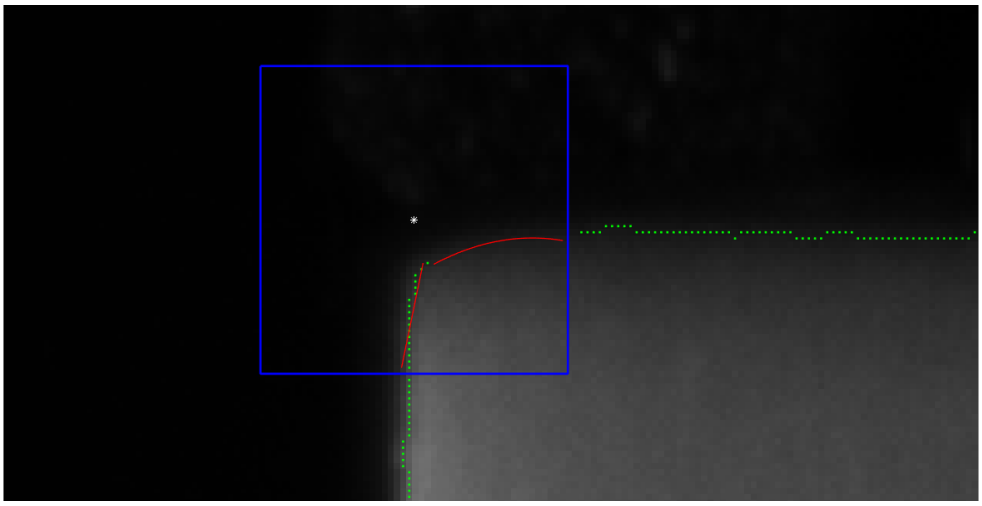


Figura 3.16: Exemplo representativo da estimação de parábolas no vértice.

Finalmente, a recta é substituída por uma segunda parábola que irá ocupar a zona lateral.

Estando desenhadas ambas as parábolas, procede-se ao cálculo das distâncias entre cada ponto do contorno e o ponto da parábola mais próximo.

A parábola pode ser definida pela Equação 3.2 e sendo conhecidos os seus parâmetros, a distância de um ponto à parábola pode ser definida pela Equação 3.3. Por fim, igualando a derivada das distâncias a zero, obtemos o ponto da parábola mais próximo do ponto de referência, possibilitando o cálculo da distância. O código pode ser consultado em *dist_para.m* no Anexo A.

$$f(x) = ax^2 + bx + c \tag{3.2}$$

$$d_i(x) = \sqrt{(x - x_0)^2 + (f(x) - y_0)^2} \tag{3.3}$$

$$\frac{dd_i(x)}{dx} = 0 \tag{3.4}$$

Onde:

(x_0, y_0) = coordenadas de referência

Embora funcional, devido à multiplicidade de elementos presentes na região e falta de implementação de processos de cálculo vectorial, este procedimento torna-se um factor limitante do desempenho do algoritmo por causa do elevado custo computacional.

Deste modo, foi implementada uma permutação do processo descrito anteriormente, em que apenas se calcula a distância num dos eixos (*dist_para_dy.m* em Anexo A). O cálculo das distâncias passará a ser realizado segundo a Equação 3.5.

$$d_i(x) = f(x) - y_0 \tag{3.5}$$

O vector das distâncias calculado é, por fim, utilizado no cálculo da média (*pdist_mean*), desvio padrão (*pdist_std*) e *outliers* resultantes da estimação das parábolas, ou seja, pontos a uma distância superior a 1.5 pixeis da parábola mais próxima (*para_outliers_relevant*).

3.4.1.2 Orientações

Sendo que os pontos do contorno formam uma linha contínua e com espessura 1 pixel, foi possível obter as orientações individuais de cada ponto pelo algoritmo *skeletonOrientations.m* (de Wolski, 2013). Seguindo a sugestão de Wolski, foram representadas as normais para melhor visualização como pode ser visto no exemplo da Figura 3.17.

Este método usa por defeito um operador 5×5 para avaliar a posição dos pixeis vizinhos no cálculo das orientações.

A partir do vector das orientações é calculado o número de mudanças de direção através da contagem das mudanças de sinal — variável *corner_sign*.

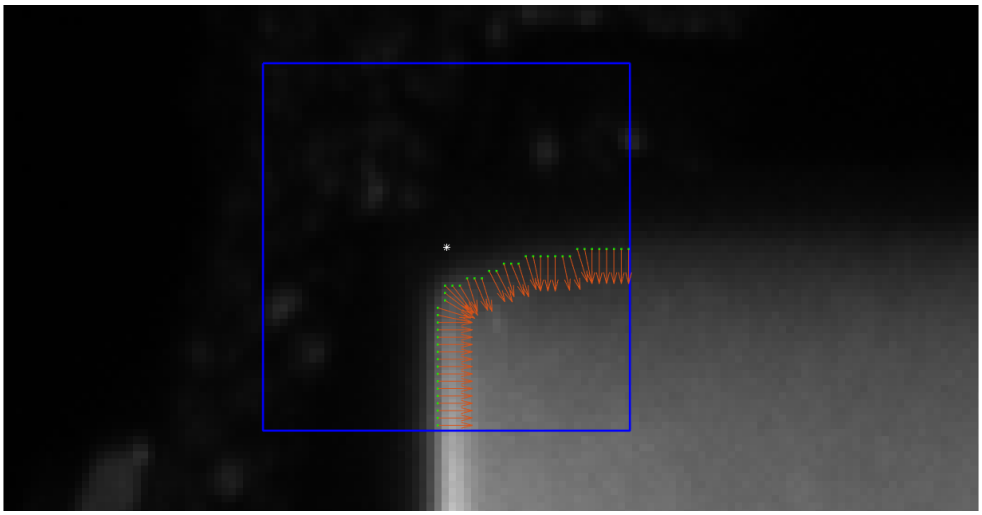


Figura 3.17: Normais das orientações obtidas pelo algoritmo *skeletonOrientations.m*.

3.4.2 Arestas

Uma aresta, no contexto do algoritmo desenvolvido neste trabalho, é constituída por um conjunto de pontos que descrevem uma trajectória idealmente linear. No entanto, as peças na realidade apresentam uma ligeira curvatura nas zonas de aproximação ao vértice. Desta forma, decidiu-se dividir a aresta em cinco zonas que serão analisadas separadamente.

Nas três zonas centrais é feita a estimação de uma recta pelo método *RANSAC*, e extraídas as distâncias de cada ponto a essa recta. Na Figura 3.18 podemos ver um exemplo desta aplicação, em que os pontos marcados a magenta representam *outliers* por estarem a uma distância da recta superior a 1 pixel.

Nas duas zonas adjacentes aos vértices, devido à ligeira curvatura presente, optou-se por fazer a estimação de uma parábola pelo método *RANSAC* de modo a acondicionar melhor os pontos da região.

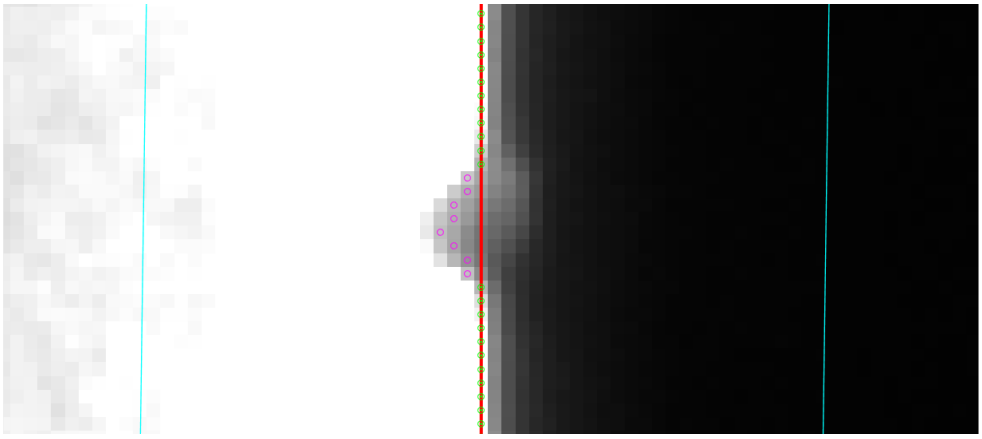


Figura 3.18: Recta estimada pelo *RANSAC* aplicado aos pontos de uma aresta com defeito.

Utilizando as funções *dist_outliers.m* e *dist_pada_dy.m* (Anexo A) são calculadas as distâncias dos pontos do contorno à recta ou parábola, a partir do qual será calculado o erro quadrático (variável *mse*), a média (*line_mean*), o desvio padrão *line_std*) e os *outliers* (line_outliers).

Utilizando o mesmo método que foi referido em 3.4.1.2 podemos obter novamente as orientações dos pontos da aresta, como representa a Figura 3.19.

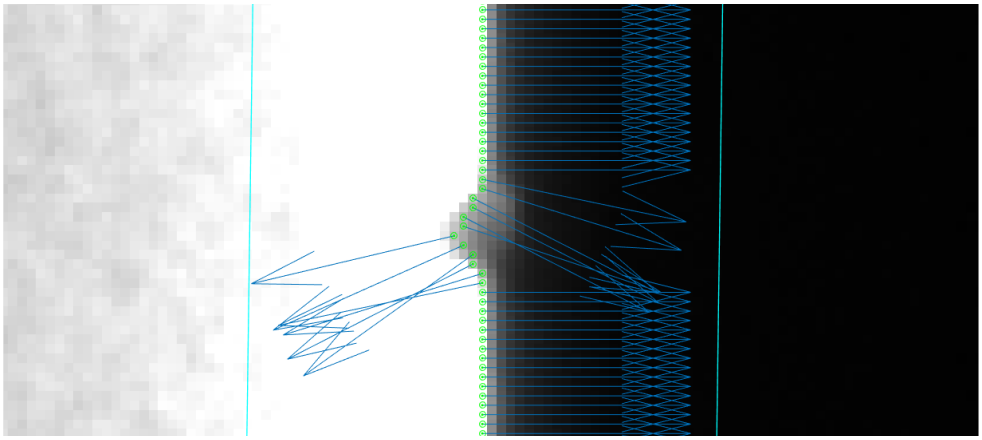


Figura 3.19: Normais das orientações obtidas pelo algoritmo *skeletonOrientations.m* numa peça defeituosa.

Seguidamente é criada uma variável chamada *line_sign* que contem o número de inflexões. Este parâmetro é calculado através das mudanças de sinal do vector das orientações.

Quando as linhas do contorno têm pior qualidade, estas tendem a assumir uma trajectória com mais ondulações, aumentando o valor do *line_sign*. Para este efeito, a função desenvolvida para calcular as inflexões pode ser aplicada com um *threshold* definido pelo utilizador (*sign_thresh.m* em Anexo A). Nesta dissertação é estudado o impacto de um *theshold* de 2 (variável *line_sign*).

3.4.3 Operador de Harris

O filtro de Harris trata-se de um operador de detecção de cantos em imagens (Harris e Stephens, 1988). Este foi o propósito com que o método foi testado neste trabalho, no entanto, após uma análise dos resultados, era evidente que este detectava com maior frequência os defeitos presentes nas arestas e não era consistente na detecção dos vértices da peça. Na realidade este fenómeno faz sentido para a maior parte dos defeitos, uma vez que estes, do ponto de vista matricial apresentam um desvio brusco de direcção do gradiente, à semelhança de um vértice.

Desta forma, este operador foi utilizado como métrica de classificação de defeitos. Para funcionar como esperado, este foi aplicado a uma versão da imagem com contraste ajustado e posteriormente intersetado com o contorno da peça para eliminar outras detecções (*rect.m* no Anexo A).

3.5 Outras funcionalidades

Além das operações do algoritmo que são directamente responsáveis pelo processamento de imagem, e portanto, fundamentais ao seu funcionamento, o algoritmo inclui funcionalidades básicas como:

- *Scan* sequencial das imagens pertencentes à base de dados especificada;
- Capacidade de criar arquivos e gravar imagens com propósito de *debug* do programa;
- Capacidade de gerar ficheiros em formato *Excel* com os resultados numéricos;
- Capacidade de criar gráficos para comparação dos resultados;
- Gerir o *output* do programa através de variáveis modificáveis (*flags*);
- Registar erros obtidos durante o funcionamento em ficheiros de texto.

3.6 Resumo

Neste capítulo foram analisados os principais mecanismos de processamento de imagem, desde a remoção de ruído, extracção do contorno da peça, selecção de regiões de interesse até à estimação de entidades geométricas e calculo de métricas.

Estas métricas serão então estudadas para percebermos a sua relevância como um indicador da presença de defeito nas peças. Estas encontram-se sintetizadas na Tabela 3.4.

Os princípios gerais de funcionamento do algoritmo encontra-se representados no diagrama da Figura 3.20.

Tabela 3.4: Tabela resumo das métricas.

Métrica	Variável	Local
Erro quadrático médio	mse	aresta
MSE das orientações	mse_orient_line	aresta
Média das distancias	line_mean	aresta
Desvio padrão distancias	line_std	aresta
Outliers *RANSAC*	line_outliers	aresta
N. Inflexões com *threshold*	line_sign	aresta
Harris	harris	aresta
Média das distancias (círculo)	circle_distout_mean	vértice
Desvio padrão distancias (círculo)	circle_distout_std	vértice
N. Inflexões	corner_sign_relevant	vértice
Raio do circulo	circle_radius_relevant	vértice
Outliers circulo	circle_outliers_relevant	vértice
Média das distâncias (parábola)	pdist_mean	vértice
Desvio padrão das distâncias (parábola)	pdist_std	vértice
Outliers das parábolas	para_outliers_relevant	vértice

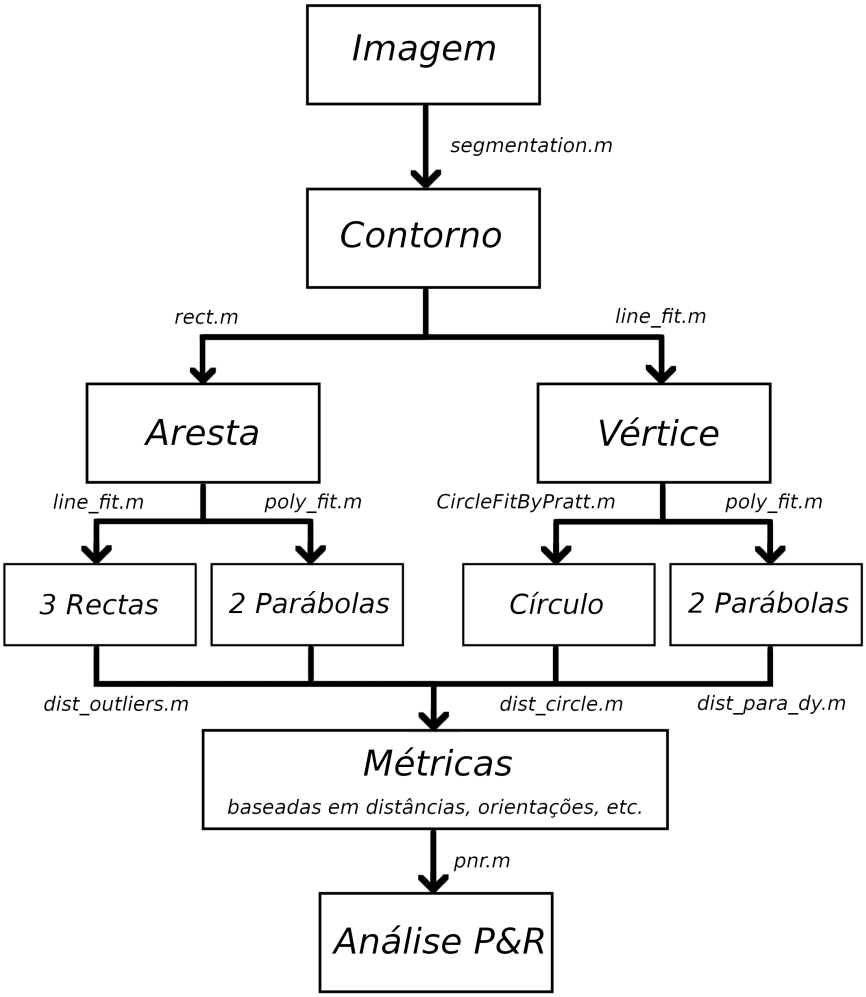


Figura 3.20: Diagrama geral do algoritmo.

Capítulo 4

Análise dos resultados obtidos

Neste capítulo será introduzido o processo utilizado na avaliação das métricas e discutidas as principais limitações encontradas.

4.1 Método utilizado

De modo a poder interpretar os resultados obtidos a partir do processamento da base de dados fornecida, recorreu-se à classificação manual das regiões das peças. Esta classificação assume a forma de dois vectores lógicos designados *defect_l* e *defect_c* em *main.m* (Anexo A) que indicam a existência ou ausência de defeito numa dada região da aresta e vértice respectivamente.

É importante considerar que a presença de defeito numa região da peça não depende exclusivamente da integridade da mesma, mas de um conjunto de factores que, além do referido anteriormente, incluem a configuração de iluminação em causa, o posicionamento e orientação da peça e o tempo de exposição luminosa.

Com estas informações, foram criadas tabelas para registo e comparação entre os valores obtidos. Exemplos destas tabelas estão disponíveis no Anexo B, Anexo C e Anexo D assim como no ficheiro *tables.xlsx* fornecido em suporte digital.

Sendo o objectivo principal do algoritmo a determinação de um método de classificação das peças em termos da existência ou ausência de defeitos nos contornos, estamos a trabalhar com um problema de classificação binária.

Deste modo é possível avaliar o desempenho das métricas calculadas dentro de uma gama de aceitação. Neste caso, a gama de aceitação será um intervalo de valores obtidos experimentalmente através da análise da representação gráfica dos resultados como será visto na Secção 4.2.

Por fim, utilizando a gama de aceitação e a classificação manual das regiões procedeu-se a uma análise de precisão e revocação de cada métrica aplicada a cada configuração luminosa estudada nesta dissertação.

4.2 Resultados obtidos

Com os dados disponíveis, procedeu-se à representação gráfica dos valores normalizados de modo a verificar a existência de separação entre os pontos correspondentes a zonas defeituosas e não defeituosas (função *do_plot.m* disponível no Anexo A).

A Figura 4.1 mostra um exemplo da variável *circle_radius_relevant* aplicada a peça número 4 sob a configuração luminosa número 3, onde os valores calculados em regiões com defeito visível estão representadas por um circulo vermelho e os correspondentes a regiões sem defeito por uma cruz azul.

Os valores foram reajustados à escala segundo a Equação 4.1 e portanto o eixo vertical da Figura 4.1 não representa os valores reais calculados do raio do circulo.

$$x'(i) = \frac{x(i) - \min(x(i))}{\max(x(i)) - \min(x(i))} \tag{4.1}$$

Onde:

$x'(i)$ = valor normalizado
$x(i)$ = valor da métrica
i = posição no vector (corresponde a uma região)

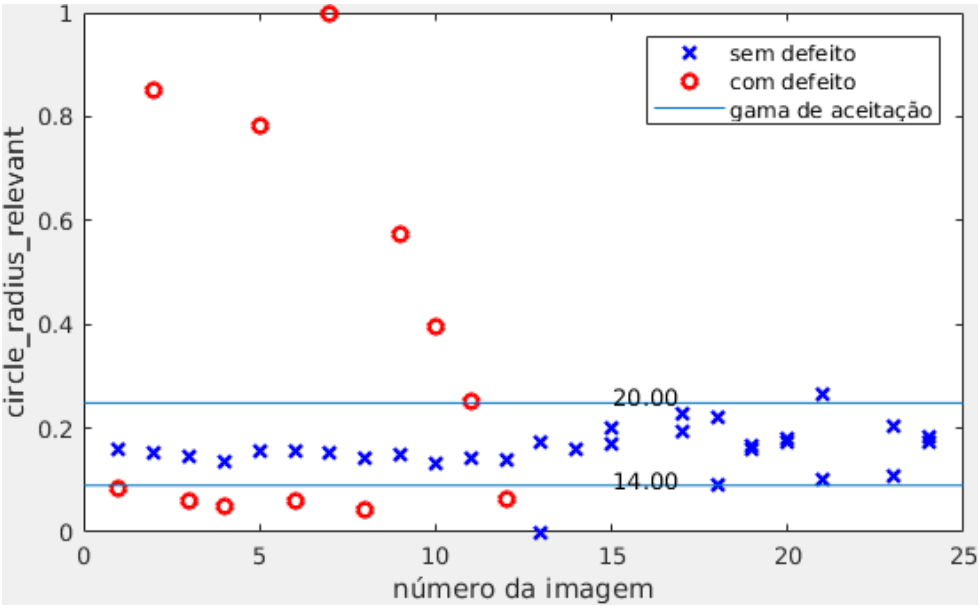


Figura 4.1: Distribuição da variável *circle_radius_relevant* de peça 4 configuração 3.

Analisando os gráficos obtidos estimou-se uma gama de aceitação para cada métrica, a partir do qual serão calculadas várias medidas representativas do seu desempenho através de uma análise de precisão e revocação (função *pnr.m* em Anexo A). As gamas de aceitação aplicadas encontram-se sintetizadas na Tabela 4.1.

Tabela 4.1: Gamas de aceitação estimadas com base nos resultados.

Métrica	Gama de aceitação
par_mse	[0 0.6]
par_mse_ori_l	[0 50]
par_lmean	[0 1]
par_lstd	[0 0.5]
par_lsign	[0 0.5]
par_lout	[0 10]
par_harris	[0 0.5]
par_cr	[14 20]
par_cs	[2 4]
par_cdmean	[0 1.3]
par_cdstd	[0 1]
par_co	[0 3]
par_pdmean	[0 0.8]
par_pdstd	[0 0.75]
par_po	[0 1]

Deste modo, seguiu-se a seguinte abordagem:

- TP — verdadeiro positivo (*true positive*) ou seja, região defeituosa classificada como defeituosa;
- FP — falso positivo (*false positive*) ou seja, região defeituosa classificada como sem defeito;
- TN — verdadeiro negativo (*true negative*) ou seja, região sem defeito classificada como defeituosa;
- FN — falso negativo (*false negative*) ou seja, região sem defeito classificada como sem defeito.

Estes parâmetros foram determinados a partir do conjunto de imagens de cada peça com a mesma configuração, apenas variando o tempo de exposição luminosa entre cada uma.

Deste modo foram calculados os valores de precisão (*ppv*), revocação (*rec*), *balanced-accuracy* (*bacc*), e especificidade (*tnr*).

4.2.1 Interpretação dos parâmetros *P&R*

Para a melhor compreensão dos resultados obtidos pela função *pnr.m* (Anexo A), segue-se a explicação e contextualização dos parâmetros calculados.

Em primeiro lugar, é de especial importância o valor da precisão (*ppv*) que representa, no contexto do problema, a percentagem de regiões com defeito que foram classificadas como defeituosas assumindo a gama de aceitação estimada. Deste modo, o *ppv* poderá ser um dos parâmetros mais fortemente correlacionado com a relevância da métrica para o processo de classificação.

A revocação (*rec*) representa a percentagem de classificações de defeito correctamente atribuídas, ou seja, a percentagem de instâncias fora da gama de aceitação que possuem defeito.

A especificidade (*tnr*) representa a percentagem de regiões sem defeito entre as classificadas como não defeituosas.

A *accuracy* corresponde à percentagem de regiões bem classificadas (com defeito e sem defeito). Como em algumas configurações a quantidade de zonas defeituosas é baixa, utilizou-se também uma adaptação deste parâmetro (*bacc*) que tem melhor comportamento em amostras desequilibradas.

4.2.2 Comentário dos resultados

Analisando os gráficos obtidos da como é exemplo o da Figura 4.2, procura-se a maior separação possível entre os pontos assinalados a vermelho dos pontos a azul, que correspondem respectivamente a regiões com defeito e sem defeito. Deste modo, seria possível escolher uma gama de valores no qual se aceita ou rejeita a peça durante o processo de inspecção automática.

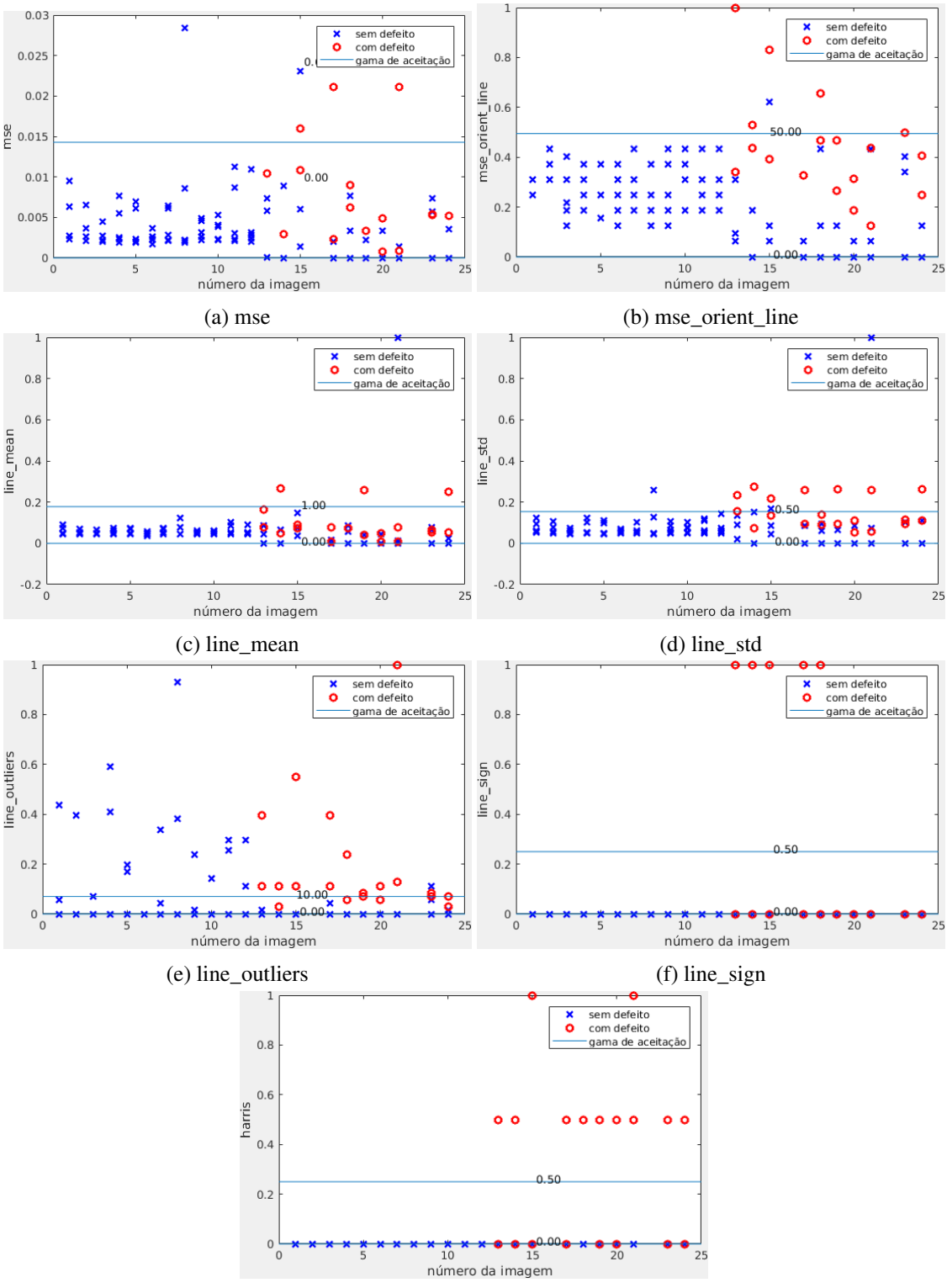
Este fenómeno foi mais evidente nos resultados obtidos do processamento das imagens com a configuração luminosa número 3, como pode ser visto na Figura 4.3. As regiões dos vértices, em particular, produziram resultados bastante satisfatórios, sendo que em todas as métricas propostas ocorre uma separação notável dos valores calculados em regiões defeituosas e sem defeito.

A única excepção a esta observação é o parâmetro *corner_sign_relevant*, que, apesar de apresentar alguma separação entre os valores representados no gráfico, não foi encontrada uma boa explicação para os resultados obtidos do ponto de vista absoluto. Estes estão mais fortemente correlacionados com limitações no *design* do algoritmo, sendo que, para o exemplo em análise, existe uma forte tendência para a peça possuir um vértice sem mudanças de orientação e os restantes com pelo menos uma mudança. Assim, não foi escolhida uma gama de aceitação que maximizasse os valores obtidos da análise de precisão e revocação porque esse intervalo não faria sentido num cenário real.

Na verdade, esta realidade prevalece sobre todas as configurações estudadas tanto para regiões vértice como aresta, apesar de apresentarem geometrias diferentes. Conclui-se portanto que a aplicação neste projecto de métricas baseadas em orientações possui menor relevância e necessitaria de mais desenvolvimento caso se pretenda estudar verdadeiramente o seu impacto.

É também de destaque o desempenho do parâmetro representativo dos *outliers* obtidos pela estimação de parábolas nos vértices — *para_outliers_relevant*. Este obteve uma classificação perfeita para a peça 4 configuração 3 como pode ser visto na Tabela 4.2 sob o nome *par_po*, no entanto este comportamento não foi replicado no caso da configuração 9 (Tabela 4.4).

Observando os dados obtidos da Tabela 4.2 e Tabela 4.3 que representam o processamento das regiões de arestas, podemos notar uma precisão (*ppv*) bastante menor comparativamente às métricas calculadas para as regiões de vértices. Este comportamento mostra-se constante em ambas as peças analisadas sob a configuração luminosa 3, o que pode significar que resulta de diversos factores.



(a) mse

(b) mse_orient_line

(c) line_mean

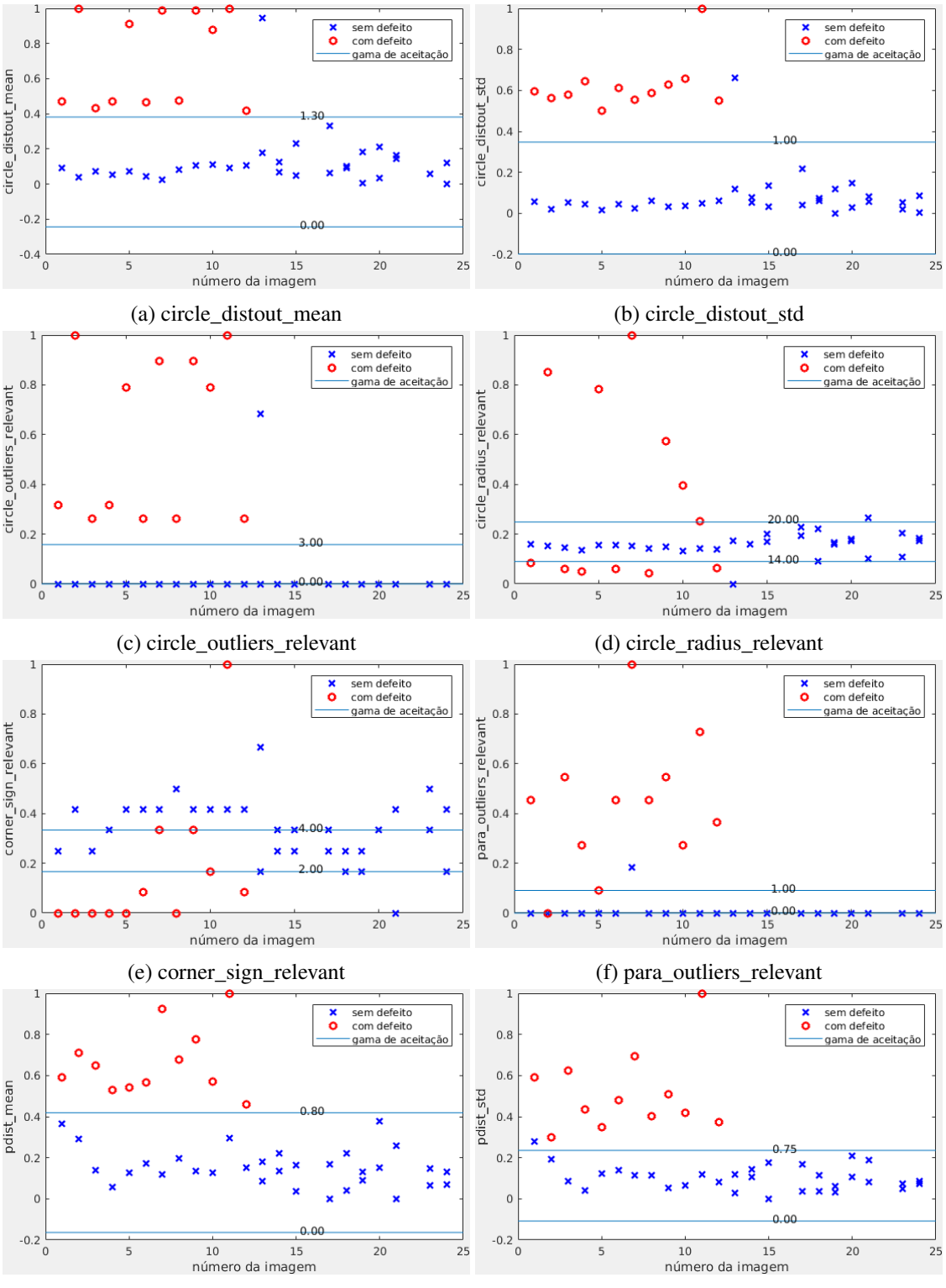
(d) line_std

(e) line_outliers

(f) line_sign

(g) harris

Figura 4.2: Representações gráficas das métricas relacionadas com o aresta da peça 4 configuração 3 e respectiva gama de aceitação selecionada.



(a) circle_distout_mean

(b) circle_distout_std

(c) circle_outliers_relevant

(d) circle_radius_relevant

(e) corner_sign_relevant

(f) para_outliers_relevant

(g) pdist_mean

(h) pdist_std

Figura 4.3: Representações gráficas das métricas relacionadas com o vértice da peça 4 configuração 3 e respectiva gama de aceitação selecionada.

Tabela 4.2: Tabela de precisão e revocação da peça 4 configuração 3.

	acc	bacc	ppv	rec	tnr
par_mse	0.86364	0.80303	0.4	0.72727	0.87879
par_mse_ori_l	0.85455	0.84455	0.25	0.83333	0.85577
par_lmean	0.86364	0.86061	0.3	0.85714	0.86408
par_lstd	0.87273	0.81888	0.45	0.75	0.88776
par_lsign	0.86364	0.92857	0.25	1	0.85714
par_lout	0.81818	0.70349	0.6	0.5	0.90698
par_harris	0.92727	0.95918	0.6	1	0.91837
par_cr	0.93182	0.9	1	0.8	1
par_cs	0.59091	0.6125	0.75	0.375	0.85
par_cdmean	0.97727	0.96154	1	0.92308	1
par_cdstd	0.97727	0.96154	1	0.92308	1
par_co	0.97727	0.96154	1	0.92308	1
par_pdmean	0.97727	0.96154	1	0.92308	1
par_pdstd	0.95455	0.92857	1	0.85714	1
par_po	1	1	1	1	1

Tabela 4.3: Tabela de precisão e revocação da peça 0 configuração 3.

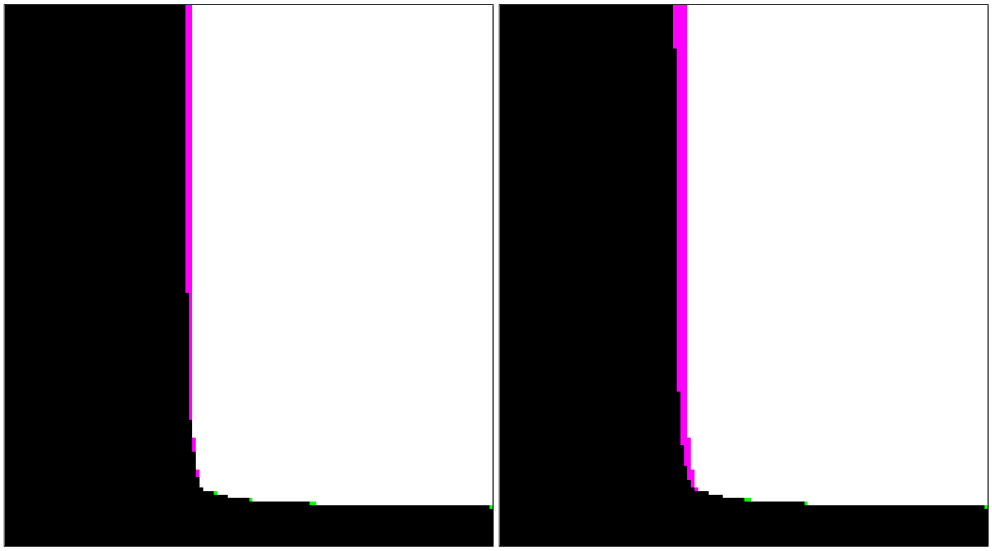
	acc	bacc	ppv	rec	tnr
par_mse	0.85714	0.92537	0.23077	1	0.85075
par_mse_ori_l	0.77143	0.40299	0	0	0.80597
par_lmean	0.85714	0.92537	0.23077	1	0.85075
par_lstd	0.85714	0.92537	0.23077	1	0.85075
par_lsign	0.81429	NaN	0	NaN	0.81429
par_lout	0.71429	0.49769	0.15385	0.18182	0.81356
par_harris	0.81429	0.67188	0.23077	0.5	0.84375

Tabela 4.4: Tabela de precisão e revocação da peça 4 configuração 9.

	acc	bacc	ppv	rec	tnr
par_mse	0.78125	0.41667	0	0	0.83333
par_mse_ori_l	0.38281	0.4506	0.45	0.11688	0.78431
par_lmean	0.82031	0.42	0	0	0.84
par_lstd	0.78125	0.50575	0.1	0.16667	0.84483
par_lsign	0.78906	0.47572	0.05	0.11111	0.84034
par_lout	0.70312	0.49849	0.2	0.15385	0.84314
par_harris	0.84375	0.6746	0.05	0.5	0.84921
par_cr	0.71875	0.37097	0	0	0.74194
par_cs	0.65625	0.71053	1	0.42105	1
par_cdmean	0.84375	0.91379	0.375	1	0.82759
par_cdstd	0.84375	0.91379	0.375	1	0.82759
par_co	0.8125	0.9	0.25	1	0.8
par_pdmean	0.71875	0.37097	0	0	0.74194
par_pdstd	0.84375	0.91379	0.375	1	0.82759
par_po	0.8125	0.9	0.25	1	0.8

Um desses factores pode ser o facto de estarmos a agrupar na mesma classificação regiões analisadas com distâncias obtidas a partir de uma recta (regiões centrais da aresta) e distâncias obtidas a partir de uma parábola (extremidades da aresta).

Também é possível que tenha influência nos resultados a decisão de agrupar as imagens cujos tempos de exposição luminosa variam entre si. Esta variação é menos acentuada na configuração 3, tomando valores entre o 1ms e 12ms, enquanto que na configuração 9 os tempos de exposição variam de 20ms a 200ms. Como podemos ver na Figura 4.4, este efeito não é desprezável e o seu impacto é mais acentuado na morfologia da aresta, afectando a sua curvatura.

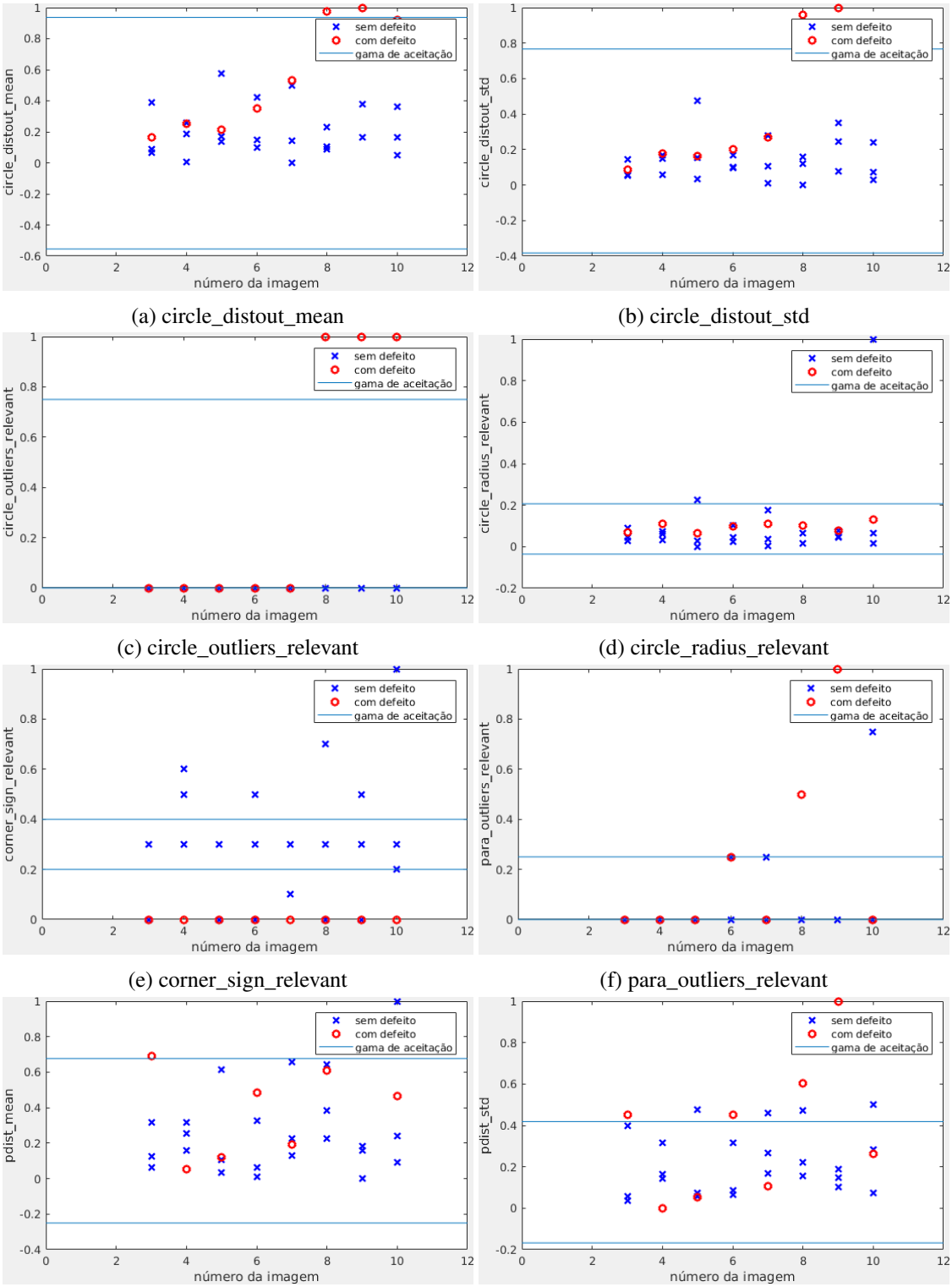


(a) 1ms - 3ms (b) 1ms - 7ms

Figura 4.4: Comparação entre diferentes tempos de exposição luminosa no contorno da peça 0 com a configuração luminosa 3, em que a zona magenta representa a diferença entre imagens com os tempos especificados.

Portanto, a grande gama de tempos de exposição luminosa pela qual a configuração 9 é composta provoca uma grande variação no comportamento da detecção de contornos e, nos extremos, erros de segmentação graves em que o processamento da imagem é abortado. Isto acontece em imagens com tempo de exposição baixo, em que o contorno da peça revela variações luminosas não relacionadas com a presença de defeitos, e com tempos de exposição muito elevados em que os defeitos são camuflados pelo excesso de iluminação. Todas estas variações de comportamento produzem resultados em que não é possível escolher gamas de aceitação funcionais, como podemos ver na Figura 4.5.

Nas configurações 4 e 6, como não existe imagens com defeito nos contornos, não será possível efetuar a análise no contexto deste capítulo, uma vez que a precisão seria de 0.



(a) circle_distout_mean

(b) circle_distout_std

(c) circle_outliers_relevant

(d) circle_radius_relevant

(e) corner_sign_relevant

(f) para_outliers_relevant

(g) pdist_mean

(h) pdist_std

Figura 4.5: Representações gráficas das métricas relacionadas com o vértice da peça 4 configuração 9 e respectiva gama de aceitação selecionada.

4.2.3 Limitações

Para analisar os resultados obtidos é necessário ter em mente todos os factores que podem ter condicionado o funcionamento do algoritmo.

Um dos problemas foi a reduzido número de imagens relevantes ao estudo de defeitos nos contornos da peça. Apesar da base de dados fornecida inicialmente possuir uma quantidade razoável de elementos, estes encontram-se dispersos por todas as combinações de configuração luminosa, posicionamento e orientação da peça, tempos de exposição, etc. Além disso, uma parte das configurações selecionadas tem utilidade reduzida devido à ausência de defeitos nos contornos, e portanto, a escassez de regiões defeituosas torna a base da dados desequilibrada.

Deste modo, podemos dizer que os resultados são baseados num número reduzido de evidências e para considerar a sua implementação em ambiente industrial, teriam de ser realizados testes mais profundos tentando replicar os valores obtidos.

Como já foi mencionado na Subsecção 4.2.2, para o cálculo dos parâmetros de precisão e revocação foram agrupadas as imagens com diferentes tempos de exposição o que pode ter um impacto significativo nos valores obtidos, afectando principalmente as regiões das arestas.

Ao nível do *setup* da captura das imagens existem também algumas limitações, como por exemplo pequenos desvios na orientação da peça, não ficando perfeitamente vertical. Isto deve-se provavelmente a folgas existentes nos componentes mecânicos, que são responsáveis pelo posicionamento automático e fixação do objecto durante o processo de captura. Este fenómeno tem um impacto considerável em configurações com incidência lateral da luz, resultando numa distribuição da luminosidade não uniforme ao longo da aresta relevante, e portanto criando uma assimetria que afeta o processamento da imagem.

4.2.4 Tempos de processamento

Durante o funcionamento do algoritmo, este regista os tempos que demora a realizar cada uma das etapas principais. Tendo em conta as características do computador utilizado que se encontram sintetizadas na Tabela 4.5, o tempo que demora em média para processar cada imagem encontra-se descrito na Tabela 4.6.

A Figura 4.6 compara o tempo de processamento entre a configuração 3, que analisa um lado da peça em cada imagem, da configuração 9 onde todos os lados são analisados simultâneamente. Podemos notar que a principal diferença dá-se na função *rect.m* que é responsável por todas as operações de processamento de arestas. Também podemos reparar que as variáveis relacionadas com o vértice têm aproximadamente o mesmo valor, devendo-se este fenómeno ao facto de as configurações unilaterais calculam os parâmetros dos quatro vértices e apenas selecionam dois no final. Isto trata-se portanto de uma ineficiência na forma como o programa foi desenvolvido que existe por motivos de compatibilidade, mas que, naturalmente, pode ser melhorado em versões futuras.

Tabela 4.5: Especificações do computador utilizado no desenvolvimento do algoritmo.

Componente	Modelo
OS	Arch Linux x86_64
Kernel	5.2.9-arch1-1-ARCH
CPU	Intel i7-4710HQ (8) @ 2.500GHz
RAM	16GB DDR3 @ 1600MHz

Tabela 4.6: Tempos de processamento.

Configuração	Tempo total
Configuração 3	11.626s
Configuração 4	13.312s
Configuração 6	18.097s
Configuração 9	18.810s

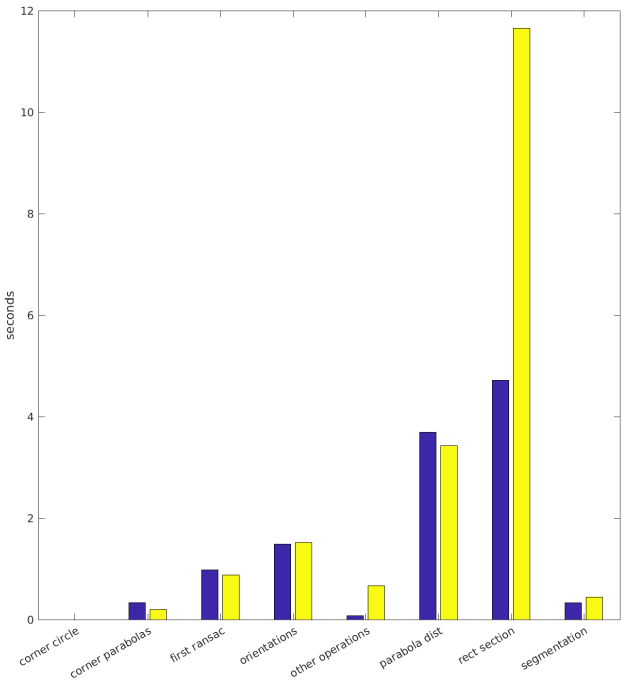


Figura 4.6: Tempos de processamento da peça 4 configuração 3 (azul) e configuração 9 (amarelo).

4.3 Resumo

Neste capítulo foi descrito o processo utilizado na interpretação dos resultados, desde a criação de gráficos para analisar a dispersão dos valores, à abordagem tomada no cálculo de parâmetros que são indicativos do desempenho das métricas.

Foram também discutidos os resultados da análise de precisão e revocação para as diferentes configurações estudadas, e por fim, as limitações que podem ter influência nos valores obtidos.

Deste modo, concluiu-se que a configuração 3 obteve o melhor desempenho, especialmente nas métricas associadas à análise de defeitos nos vértices. As métricas relacionadas com a análise de arestas tiveram todas uma precisão abaixo do que foi observado na análise do vértice, no entanto demonstram um comportamento satisfatório nos restantes parâmetros.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Satisfação dos Objetivos

Nesta dissertação procurou-se construir um algoritmo de processamento de imagem aplicado ao caso específico de um puxador em alumínio. Este teria de ser capaz de implementar diversos métodos de análise com o objectivo de melhorar a nossa compreensão do problema e sobre o método mais eficaz de classificar o objecto quanto à presença de defeitos superficiais. Sendo assim, optou-se por analisar apenas os defeitos presentes no contorno da face frontal da peça, uma vez que defeitos localizados no interior da face requerem uma estratégia de processamento diferente.

Para poder classificar a integridade da peça a partir das diferentes abordagens utilizadas, há a necessidade da criação de métricas baseadas em diferentes relações geométricas entre as entidades que constituem o plano de processamento. Uma vez desenvolvido o funcionamento básico do algoritmo, tornou-se necessário criar uma compreensão global do seu desempenho. Isto foi feito através de uma análise de precisão e revocação, calculando vários parâmetros representativos da relevância de cada métrica como elemento de classificação das peças.

Deste trabalho resultou então um algoritmo com as funcionalidades descritas, e que para as configurações 3 e 9, que correspondem respectivamente a um esquema de iluminação com barra LED lateral e em *dark field ring* respectivamente, produz resultados relevantes, quer do ponto de vista de identificar bons métodos de análise, quer do ponto de vista de iluminar problemas inerentes a um projecto deste tipo.

Verificou-se deste modo uma maior facilidade em detectar defeitos nos vértices da peça quer através da estimação de um círculo, quer através da estimação de duas parábolas. Neste caso, o raio do círculo, desvio padrão das distâncias e cálculo de *outliers* mostraram, na amostra utilizada, potencial como métodos de classificação da peça.

Na análise de defeitos presentes na aresta da peça, os resultados obtidos nesta dissertação foram menos conclusivos devido a uma série de limitações discutidas na Subsecção 4.2.3. Entre as métricas calculadas, a que mostrou resultados mais interessantes foi o número de detecções do

filtro de Harris na aresta, que apesar de uma precisão inferior ao observado nos métodos aplicados no vértice, não houve ocorrência de falsas detecções.

5.2 Trabalho Futuro

Esta dissertação aborda apenas aspectos relativos a uma pequena parte dos defeitos que podem ser encontrados num problema deste tipo. Deste modo, para desenvolver uma implementação de visão artificial eficaz e completa, pode ser explorado em trabalhos futuros:

- A replicação dos métodos utilizados noutras faces no objecto, efetuando as permutações necessárias ao bom funcionamento do programa;
- A criação de um método de análise de outro tipo de defeitos presentes no objecto, nomeadamente os defeitos situados no interior das faces, fazendo uso das configurações luminosas presentes na base de dados que não tiveram relevância nesta dissertação;
- O melhoramento da base de dados atual, aumentando a sua dimensão e a variedade de defeitos presentes nos exemplos fornecidos e/ou corrigindo aspectos do suporte mecânico para o melhor posicionamento da peça;
- Se realizado o tópico anterior, explorar a possibilidade de treinar sistemas de *machine learning*, como por exemplo SVMs (*support-vector machines*) na classificação automática das peças.

Referências

John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. PAMI-8*, páginas 679–698, novembro 1986.

Sean de Wolski. skeletonorientation.m, 2013. Disponível em https://www.mathworks.com/matlabcentral/answers/88714-how-to-find-the-direction-angle-of-points-on-an-edge-in-a-digital-image, acedido a última vez em 25 de junho de 2019.

Martin A. Fischler e Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM Vol 24*, páginas 381–395, junho 1981.

Rafael C. Gonzalez e Richard E. Woods. *Digital Image Processing*. Prentice Hall, Second edição, 2002. ISBN 0-201-18075-8.

Chris Harris e Mike Stephens. A combined corner and edge detector. Relatório técnico, Plessey Research Roke Manor, 1988.

C. Iglesias, J. Martínez e J. Taboada. Automated vision system for quality inspection of slate slabs. *Computers in Industry*, páginas 119–129, março 2018.

Raman Maini e Dr. Himanshu Aggarwal. Study and comparison of various image edge detection techniques, 2009.

João Martins. Visão artificial na inspecção e caracterização de defeitos superficiais em peças fundidas, outubro 2018.

Mathworks. Detect cell using edge detection and morphology, 2019. Disponível em https://www.mathworks.com/help/images/detecting-a-cell-using-image-segmentation.html, acedido a última vez em 04 de setembro de 2019.

MATLAB. *MATLAB:Computer Vision Toolbox Reference*. The MathWorks Inc., Natick, Massachusetts, 2019.

Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics Vol 9*, páginas 62–66, janeiro 1979.

V. Pratt. Direct least-squares fitting of algebraic surfaces. *Computer Graphics Vol.21*, páginas 145–152, 1987.

Prewitt. Object enhancement and extraction, 1970.

L. Roberts. Machine perception of 3-d solids, optical and electro-optical information processing. *MIT Press*, 1965.

E. Sobel. Camera models and machine percetion, 1970.

Chris Solomon e Toby Breckon. *Fundamentals of Digital Image Processing*. Wiley-Blackwell, First edição, 2011. ISBN 978-0-470-84472-4.

Anexo A

Código Matlab

```
%% main script

%prepare worksapce (bugs without it for now)
clear;
clc;
close all;

%flags
histo_flag = 0;
debug_flag = 0;
plot_flag = 1;

if debug_flag == 0
    plot_flag = 0;
end

% -- CHANGE PATH TO BEFORE RUN --
addpath /home/<user>/Documents/Insp_Visual(JT)/ctorres_sta/export_fig;
addpath /home/<user>/Documents/Insp_Visual(JT)/ctorres_sta/circlefit;

%supress irrelevant warnings
warning('off','MATLAB:polyfit:RepeatedPointsOrRescale')
warning('off','vision:ransac:maxTrialsReached')
warning('off','images:initSize:adjustingMag')
warning('off','MATLAB:MKDIR:DirectoryExists')

%make missing directories
mkdir output;
mkdir output_seg;

database = '/home/<user>/Documents/Insp_Visual(JT)/STA - DB - Joao ...
    Martins/';

```

```
33 %define which configs to scan
34 frente = dir(fullfile(database,'*frente_config_3'));
35 frente = [frente; dir(fullfile(database,'*frente_config_4'))]; %too ...
       many errors
36 frente = [frente; dir(fullfile(database,'*frente_config_5'))];
37 frente = [frente; dir(fullfile(database,'*frente_config_6'))];
38 frente = [frente; dir(fullfile(database,'*frente_config_9'))];
39 %frente = [frente; dir(fullfile(database,'*frente_config_10'))];
40
41 lado = dir(fullfile(database,'*lado_e_config_3'));
42 lado = [lado; dir(fullfile(database,'*lado_e_config_4.e'))];
43 lado = [lado; dir(fullfile(database,'*lado_e_config_5'))];
44 lado = [lado; dir(fullfile(database,'*lado_e_config_6'))];
45 lado = [lado; dir(fullfile(database,'*lado_e_config_9'))];
46
47 folder = [frente; lado];
48
49 if debug_flag == 1
50 f1 = figure('visible','on');
51 f2 = figure('visible','on');
52 end
53
54 %create empty cell array for speed
55 t = cell(numel(frente),40);
56 orient_corner = cell(numel(frente),40);
57 orient_corner1 = cell(numel(frente),40);
58 corners = cell(numel(frente),40);
59 circle_distout = cell(numel(frente),40,4);
60 circle_distout1 = cell(numel(frente),40,4);
61 circle_distout_mean = cell(numel(frente),40,2);
62 circle_distout_std = cell(numel(frente),40,2);
63 circle_radius = cell(numel(frente),40,4);
64 circle_radius1 = cell(numel(frente),40,4);
65 circle_radius_relevant = cell(numel(frente),40,2);
66 idc = cell(numel(frente),40,2);
67 S = cell(numel(frente));
68 corner_sign = cell(numel(frente),40,4);
69 corner_sign1 = cell(numel(frente),40,4);
70 corner_sign_relevant = cell(numel(frente),40,2);
71 TL = cell(3,1);
72 TC = cell(3,1);
73 circle_outliers = cell(numel(frente),40,4);
74 circle_outliers1 = cell(numel(frente),40,4);
75 circle_outliers_relevant = cell(numel(frente),40,2);
76 pdist = cell(numel(frente),40,4);
77 pdist1 = cell(numel(frente),40,4);
78 pdist_mean = cell(numel(frente),40,2);
79 pdist_std = cell(numel(frente),40,2);
80 para_outliers = cell(numel(frente),40,4);
```

```
81  para_outliers1 = cell(numel(frente),40,4);
82  para_outliers_relevant = cell(numel(frente),40,2);
83
84  mse = cell(numel(frente),40);
85  harris = cell(numel(frente),40);
86  orient_line = cell(numel(frente),40);
87  line_sign = cell(numel(frente),40);
88  line_mean = cell(numel(frente),40);
89  line_std = cell(numel(frente),40);
90  line_outliers = cell(numel(frente),40);
91  mse_orient_line = cell(numel(frente),40);
92  relevantPts = cell(numel(frente),40);
93  distout = cell(numel(frente),40);
94  relevantOutliers = cell(numel(frente),40);
95  w_value = cell(numel(frente),1);
96
97
98  %histogram scale
99  dedge = 0:0.25:8;
100 oedge = 0:2:90;
101 cedge = 0:0.25:10;
102
103 % -- NO IDENTATION ON IMAGE SCAN LOOP TO AVOID CLIPING --- %
104
105 % begin loop
106 %for k = numel(frente)+1:numel(folder) %scan lado only
107 for k = 1:numel(frente) %scan frente only
108 % -- for k = 1:numel(folder) %scan all
109
110 folder_path = strcat(database,folder(k).name);
111 mkdir(['output/',folder(k).name]);
112 mkdir(['output_samp/',folder(k).name]);
113 mkdir(['output_hist/',folder(k).name]);
114 mkdir(['output_seg/',folder(k).name]);
115 S{k} = dir(fullfile(folder_path,'*.png'));
116
117 for u = 1:numel(S{k})
118
119 tstart = tic;
120
121 err = 1;
122
123 while err ~= 0
124 try
125 if debug_flag == 1
126 figure(f2);
127 end
128
129 F = fullfile(folder_path,S{k}(u).name);
```

```
130 I = imread(F);
131
132 % skip misplaced pictures
133 if startsWith(S{k}(u).name,'3') == 1
134     %fprintf('skip\n');
135     err = 3;
136     error('skipping')
137     %break;
138
139 end
140
141 if debug_flag == 1
142 clf;
143 end
144
145 fudge = 0.7 - err*0.2;
146
147 tic
148 %get inner line
149 if contains(folder(k).name,'config_9') == 1
150     %adicional seg for domo configs
151     %[¬, BWao] = segmentation(I,fudge, 120000, debug_flag);
152     %BWoutline = segmentation2(I, BWao);
153
154     [BWoutline, BWao] = segmentation9(I, fudge,120000, debug_flag);
155 else
156     [BWoutline, BWao] = segmentation(I, fudge, 120000, debug_flag);
157     %I(¬BWao) = 0;
158 end
159
160 t{k,u}(1) = toc;
161
162 if debug_flag == 1
163 %save image for debug
164 export_fig(fullfile(['output_seg/',folder(k).name], ...
        S{k}(u).name),'-png','-r96');
165 end
166 fprintf('seg %s\n',S{k}(u).name);
167
168 %check detected edge
169 BW = imbinarize(I);
170 % -- BWline = BW.*BWoutline.*imbinarize(Gmagx);
171
172 if debug_flag == 1
173 figure(f1);
174 imshow(I);
175 hold on;
176 text(250, 100, sprintf('fudge = %f', fudge), 'color', 'r');
177 end
```

```
178
179 tic
180 %orientations
181 orient = skeletonOrientation(BWoutline,3); %5x5 box
182
183 Onormal = orient+90; %easier to view normals
184 Onr = sind(Onormal); %vv
185 Onc = cosd(Onormal); %uu
186 [r,c] = find(BWoutline);    %row/cols
187 idx = find(BWoutline);      %Linear indices into Onr/Onc
188 %Overlay normals to verify
189 if debug_flag == 1
190 quiver(c,r,-Onc(idx),Onr(idx));
191 end
192 t{k,u}(2) = toc;
193
194 %get and plot points from edge
195 [y, x] = find(BWoutline);
196 %plot(x,y,'.')
197
198 points.S{k}(u).name = [x y];
199 sampleSize = 5; % number of points to sample per trial
200 maxDistance = 3; % max allowable distance for inliers
201 [y,x] = find(BW);
202
203 if k ≤ numel(frente)
204
205     pointsCopy = points.S{k}(u).name;
206     pointsCopy2 = points.S{k}(u).name;
207
208     %dilate bw image if thin edge
209     if contains(folder(k).name,'config_5') == 1
210         se0 = strel('line',10,0);
211         BW = imdilate(BW, se0);
212     end
213
214     tic
215     if contains(folder(k).name,'config_9') == 1 || ...
            contains(folder(k).name,'config_6') == 1 || ...
            contains(folder(k).name,'config_10') == 1
216         %just draw 2 vertical lines
217         [pointsCopy, line1v] = line_fit(pointsCopy,2, sampleSize, ...
                maxDistance, 1, debug_flag);
218     else
219         %draw 2 vertical lines and choose the brightest
220         [pointsCopy, line1v, brightIdx] = line_fit(pointsCopy,2, ...
                sampleSize, maxDistance, 1, debug_flag, BW);
221     end
222     %draw 2 horizontal lines
```

```
223     [pointsCopy, line1h] = line_fit(pointsCopy,2, sampleSize, ...
            maxDistance, 0, debug_flag);
224     t{k,u}(3) = toc;
225
226     outliers.S{k}(u).name = pointsCopy;
227
228     %prelocate for speed
229     rcx = cell(2,2);
230     rcy = cell(2,2);
231     corner_tmp = zeros(4,2);
232     ci = 0;
233     corner_di = 25;
234
235     %corners section (4)
236     for j1=1:2
237         for j2=1:2
238             %fit linear polynomial
239             p1 = polyfit(line1v{j1}(:,1),line1v{j1}(:,2),1);
240             p2 = polyfit(line1h{j2}(:,1),line1h{j2}(:,2),1);
241             %calculate intersection
242             x_intersect = fzero(@(x) polyval(p1-p2,x),3);
243             y_intersect = polyval(p1,x_intersect);
244             if debug_flag == 1
245             plot(x_intersect,y_intersect,'w*')
246             end
247             ci = ci +1;
248             corner_tmp(ci,:) = [x_intersect y_intersect];
249
250             %draw rectangles for the corners
251             rcx{j1,j2} = [x_intersect-corner_di x_intersect+corner_di ...
                    x_intersect+corner_di x_intersect-corner_di ...
                    x_intersect-corner_di];
252             rcy{j1,j2} = [y_intersect-corner_di y_intersect-corner_di ...
                    y_intersect+corner_di y_intersect+corner_di ...
                    y_intersect-corner_di];
253
254             %clean outliers vector
255             in = inpolygon(...
256                 outliers.S{k}(u).name(:,1),...
257                 outliers.S{k}(u).name(:,2),...
258                 rcx{j1,j2},rcy{j1,j2});
259             outliers.S{k}(u).name(in,:) = [];
260
261             %clean points vector
262             in = inpolygon(...
263                 pointsCopy2(:,1),pointsCopy2(:,2),...
264                 rcx{j1,j2},rcy{j1,j2});
265             cornerPts = pointsCopy2(in,:);
266             pointsCopy2(in,:) = [];
```

```
267
268             if debug_flag == 1
269             plot(rcx{j1,j2},rcy{j1,j2},'b','linewidth',2)
270             end
271
272             %set cornerPts orientations
273             orient_corner1{k,u,ci} = orient(...
274                 sub2ind(size(I),cornerPts(:,2),cornerPts(:,1)));
275
276             %changes in sign
277             corner_sign1{k,u,ci} = numel(...
278                 find(abs(diff(sign(orient_corner1{k,u,ci}),1)) == 2));
279
280             cornerPts_copy = cornerPts;
281             cornerPts_copy2 = cornerPts;
282
283             %circle scraps
284             tic
285             par = CircleFitByPratt(cornerPts);
286             t{k,u}(3+ci) = toc;
287             if debug_flag == 1
288             viscircles([par(1) par(2)], par(3));
289             end
290
291             %radius
292             circle_radius1{k,u,ci} = par(3);
293
294             %distances
295             circle_distout1{k,u,ci} = dist_circle(cornerPts,par);
296
297             idd = find(circle_distout1{k,u,ci}>3);
298             %outliers
299             circle_outliers1{k,u,ci} = numel(circle_distout1{k,u,ci}(idd));
300
301             %% -- parabola section -- %%
302             tic
303             %draw line
304             cornerPts = line_fit(cornerPts,1,sampleSize, maxDistance, ...
                    1, debug_flag);
305
306             %draw parabola
307             [¬, inlierPts, p_coef1] = poly_fit(cornerPts, 1, ...
                    maxDistance, 0, debug_flag);
308
309             [¬, idInt] = intersect(cornerPts_copy, inlierPts,'rows');
310             cornerPts_copy(idInt,:) = [];
311
312             %draw parabola
```

```
313             [cornerPts_copy,¬, p_coef2] = poly_fit(cornerPts_copy, 1, ...
                    maxDistance, 1, debug_flag);
314
315             t{k,u}(7+ci) = toc;
316
317             %plot corner outliers
318             if debug_flag == 1
319             plot(cornerPts_copy(:,1),cornerPts_copy(:,2),'yo')
320             end
321
322             %para outliers
323             para_outliers1{k,u,ci} = numel(cornerPts_copy(:,1));
324
325             tic
326             dist1 = dist_para_dy(cornerPts_copy2, p_coef1);
327             %swap x and y
328             dist2 = dist_para_dy([cornerPts_copy2(:,2) ...
                    cornerPts_copy2(:,1)], p_coef2);
329
330             pdist1{k,u,ci} = min(dist1, dist2);
331
332             t{k,u}(11+ci) = toc;
333         end
334     end
335
336     corners{k,u} = corner_tmp;
337
338     orient_corner_tmp = cell(4);
339     corner_sign_tmp = cell(4);
340     circle_radius_tmp = cell(4);
341     circle_distout_tmp = cell(4);
342     circle_outliers_tmp = cell(4);
343     pdist_tmp = cell(4);
344     para_outliers_tmp = cell(4);
345
346     %%order corners for the domo config
347     %sort y
348     [¬, id] = sort(corners{k,u}(:,2));
349     corners{k,u} = corners{k,u}(id,:);
350     for i=1:4
351     orient_corner_tmp{i} = orient_corner1{k,u,id(i)};
352     corner_sign_tmp{i} = corner_sign1{k,u,id(i)};
353     circle_radius_tmp{i} = circle_radius1{k,u,id(i)};
354     circle_distout_tmp{i} = circle_distout1{k,u,id(i)};
355     circle_outliers_tmp{i} = circle_outliers1{k,u,id(i)};
356     pdist_tmp{i} = pdist1{k,u,id(i)};
357     para_outliers_tmp{i} = para_outliers1{k,u,id(i)};
358     end
359     %sort x
```

```
360     [¬, id] = sort(corners{k,u}(1:2,1));
361     corners{k,u}(1:2,:) = corners{k,u}(id,:);
362     for i=1:2
363     orient_corner{k,u,i} = orient_corner_tmp{id(i)};
364     corner_sign{k,u,i} = corner_sign_tmp{id(i)};
365     circle_radius{k,u,i} = circle_radius_tmp{id(i)};
366     circle_distout{k,u,i} = circle_distout_tmp{id(i)};
367     circle_outliers{k,u,i} = circle_outliers_tmp{id(i)};
368     pdist{k,u,i} = pdist_tmp{id(i)};
369     para_outliers{k,u,i} = para_outliers_tmp{id(i)};
370     end
371     %sort x
372     [¬, id] = sort(corners{k,u}(3:4,1), 'descend');
373     corners{k,u}(3:4,:) = corners{k,u}(plus(id,2),:);
374     for i=1:2
375     orient_corner{k,u,i+2} = orient_corner_tmp{id(i)+2};
376     corner_sign{k,u,i+2} = corner_sign_tmp{id(i)+2};
377     circle_radius{k,u,i+2} = circle_radius_tmp{id(i)+2};
378     circle_distout{k,u,i+2} = circle_distout_tmp{id(i)+2};
379     circle_outliers{k,u,i+2} = circle_outliers_tmp{id(i)+2};
380     pdist{k,u,i+2} = pdist_tmp{id(i)+2};
381     para_outliers{k,u,i+2} = para_outliers_tmp{id(i)+2};
382     end
383
384     %label corners
385     if debug_flag == 1
386     for i=1:4
387     text(corners{k,u}(i,1)+50,corners{k,u}(i,2)+50, int2str(i), ...
            'color', 'r')
388     end
389     end
390
391     %relevant rectangles
392     %get all sides (config 9, 6 and 10)
393     if contains(folder(k).name,'config_9') == 1 || ...
            contains(folder(k).name,'config_6') == 1 || ...
            contains(folder(k).name,'config_10') == 1
394
395         %line zones
396         wu = 16;
397
398         %ensure same order of detected lines
399         if line1v{2}(1,1) < line1v{1}(1,1)
400             lineaux = line1v{2};
401             line1v{2} = line1v{1};
402             line1v{1} = lineaux;
403         end
404         if line1h{2}(1,2) < line1h{1}(1,2)
405             lineaux = line1h{2};
```

```
406             line1h{2} = line1h{1};
407             line1h{1} = lineaux;
408         end
409
410         if debug_flag==1
411         for i=1:2
412         text(line1v{i}(1,1)+10,line1v{i}(1,2)+50, int2str(i), 'color', 'g')
413         text(line1h{i}(1,1)+50,line1h{i}(1,2)+10, int2str(i), 'color', 'g')
414         end
415         end
416
417         %pre-alocate space for speed
418         distout_v = cell(2,1);
419         distout_h = cell(2,1);
420         mse_v = cell(2,1);
421         mse_h = cell(2,1);
422         harris_v = cell(2,1);
423         harris_h = cell(2,1);
424         relevantPts_v = cell(2,1);
425         relevantPts_h = cell(2,1);
426         relevantOutliers_v = cell(2,1);
427         relevantOutliers_h = cell(2,1);
428
429         outliersN = numel(outliers.S{k}(u).name(:,1));
430
431         tic
432         for j = 1:2 %2 times
433             %draw 2 vertical relevant rectangles
434             [relevantPts_v{j}, relevantOutliers_v{j}, distout_v{j}, ...
                    mse_v{j}, line_rect, ¬,harris_v{j}] = rect(pointsCopy2, ...
                    line1v{j}, 1, corners{k,u},I,BWoutline,debug_flag);
435             %draw 2 horizontal relevant rectangles
436             [relevantPts_h{j}, relevantOutliers_h{j}, distout_h{j}, ...
                    mse_h{j},¬,¬,harris_h{j}] = rect(pointsCopy2, line1h{j}, ...
                    0, corners{k,u},I,BWoutline,debug_flag);
437         end
438         t{k,u}(16) = toc;
439
440         mse{k,u} = [mse_v{1} mse_v{2} mse_h{1} mse_h{2}];
441         harris{k,u} = [harris_v{1} harris_v{2} harris_h{1} harris_h{2}];
442         distout{k,u} = [distout_v{1} distout_v{2} distout_h{1} ...
                distout_h{2}];
443
444         counter1 = 0;
445         counter2 = 0;
446         %line metrics (16 zones)
447         for j=1:2
448             for i=1:5
449                 counter1 = counter1 + 1;
```

```
450                 line_mean{k,u}(counter1) = mean(distout_v{j}{i});
451                 line_std{k,u}(counter1) = std(distout_v{j}{i});
452                 orient_line{k,u}{counter1} = orient(...
453                     sub2ind(size(I),relevantPts_v{j}{i}(:,2),...
454                     relevantPts_v{j}{i}(:,1)));
455                 n = numel(orient_line{k,u}{counter1});
456                 mse_orient_line{k,u}(counter1) = ...
                        1/n*sum(minus(abs(orient_line{k,u}{counter1}),90).^2);
457                 line_outliers{k,u}(counter1) = ...
                        numel(relevantOutliers_v{j}{i});
458                 line_sign{k,u}(counter1) = ...
                        sign_thresh(orient_line{k,u}{counter1}',2);
459             end
460             for i=1:3
461                 counter2 = counter2 + 1;
462                 line_mean{k,u}(counter2+10) = mean(distout_h{j}{i});
463                 line_std{k,u}(counter2+10) = std(distout_h{j}{i});
464                 orient_line{k,u}{counter2+10} = orient(...
465                     sub2ind(size(I),relevantPts_h{j}{i}(:,2),...
466                     relevantPts_h{j}{i}(:,1)));
467                 n = numel(orient_line{k,u}{counter2+10});
468                 line_sign{k,u}(counter2+10) = ...
                        sign_thresh(orient_line{k,u}{counter2+10}',2);
469                 mse_orient_line{k,u}(counter2+10) = ...
                        1/n*sum(minus(abs(orient_line{k,u}{counter2+10}),0).^2);
470                 line_outliers{k,u}(counter2+10) = ...
                        numel(relevantOutliers_h{j}{i});
471                 line_sign{k,u}(counter2+10) = ...
                        sign_thresh(orient_line{k,u}{counter2+10}',2);
472             end
473         end
474
475         %corner metrics (4 zones)
476         for i=1:4
477         circle_radius_relevant{k,u}(i) = circle_radius{k,u,i};
478         corner_sign_relevant{k,u}(i) = corner_sign{k,u,i};
479         circle_distout_mean{k,u}(i) = mean(circle_distout{k,u,i});
480         circle_distout_std{k,u}(i) = std(circle_distout{k,u,i});
481         circle_outliers_relevant{k,u}(i) = circle_outliers{k,u,i};
482
483         pdist_mean{k,u}(i) = mean(pdist{k,u,i});
484         pdist_std{k,u}(i) = std(pdist{k,u,i});
485         para_outliers_relevant{k,u}(i) = para_outliers{k,u,i};
486         end
487
488         if debug_flag == 1
489         text(250, 150, sprintf('%s', folder(k).name), 'color', 'r');
```

```
490         text(250, 300, sprintf('mse1 = %f\nmse2 = %f\nmse3 = %f\nmse4 = ...
                %f\nmse5 = %f', mse{k,u}(1), mse{k,u}(2), mse{k,u}(3), ...
                mse{k,u}(4), mse{k,u}(5)), 'color', 'r');
491         text(250, 500, sprintf('circle 1 radius = %f\ncircle 2 radius = ...
                %f', circle_radius_relevant{k,u}(1), ...
                circle_radius_relevant{k,u}(2)), 'color', 'r');
492         text(250, 600, sprintf('k = %i\n u = %i', k, u), 'color', 'r');
493         end
494
495     else %get bright side only
496
497         %number of line zones
498         wu = 5;
499
500         tic
501         %normal routine
502         [relevantPts{k,u},relevantOutliers{k,u},distout{k,u},mse{k,u}, ...
                line_rect, idc, harris{k,u}] = rect(pointsCopy2, ...
                line1v{brightIdx}, 1, corners{k,u}, I,BWoutline, debug_flag);
503         t{k,u}(16) = toc;
504
505         %corner metrics (2 zones)
506         for i=1:2
507         circle_radius_relevant{k,u}(i) = circle_radius{k,u,idc(i)};
508         corner_sign_relevant{k,u}(i) = corner_sign{k,u,idc(i)};
509         circle_distout_mean{k,u}(i) = mean(circle_distout{k,u,idc(i)});
510         circle_distout_std{k,u}(i) = std(circle_distout{k,u,idc(i)});
511         circle_outliers_relevant{k,u}(i) = circle_outliers{k,u,idc(i)};
512
513         pdist_mean{k,u}(i) = mean(pdist{k,u,idc(i)});
514         pdist_std{k,u}(i) = std(pdist{k,u,idc(i)});
515         para_outliers_relevant{k,u}(i) = para_outliers{k,u,idc(i)};
516         end
517
518         %line metrics (3 zones)
519         for i=1:wu
520             line_mean{k,u}(i) = mean(distout{k,u}{i});
521             line_std{k,u}(i) = std(distout{k,u}{i});
522             orient_line{k,u}{i} = orient(...
523                 sub2ind(size(I),relevantPts{k,u}{i}(:,2),...
524                 relevantPts{k,u}{i}(:,1)));
525             n = numel(orient_line{k,u}{i});
526             mse_orient_line{k,u}(i) = ...
                    1/n*sum(minus(abs(orient_line{k,u}{i}),90).^2);
527             line_outliers{k,u}(i) = numel(relevantOutliers{k,u}{i});
528             line_sign{k,u}(i) = sign_thresh(orient_line{k,u}{i}',2);
529         end
530
```

```
531         outliersN = numel(relevantOutliers{k,u}{1}(:,1)) + ...
                numel(relevantOutliers{k,u}{2}(:,1)) + ...
                numel(relevantOutliers{k,u}{i}(:,1));
532
533         %to avoid errors
534         outliers.S{k}(u).name = [relevantOutliers{k,u}{1}; ...
                relevantOutliers{k,u}{2}; relevantOutliers{k,u}{3}];
535
536         %line_sign{k,u} = ...
                numel(find(abs(diff(sign(orient_line{k,u}),1)) == 2));
537
538         %print text on image for debug
539         if debug_flag == 1
540         text(250, 150, sprintf('%s', folder(k).name), 'color', 'r');
541         text(250, 350, sprintf('mse1 = %f\nmse2 = %f\nmse3 = %f\nmse4 = ...
                %f\nmse5 = %f', mse{k,u}(1), mse{k,u}(2), mse{k,u}(3), ...
                mse{k,u}(4), mse{k,u}(5)), 'color', 'r');
542         text(250, 500, sprintf('circle 1 radius = %f\ncircle 2 radius = ...
                %f', circle_radius_relevant{k,u}(1), ...
                circle_radius_relevant{k,u}(2)), 'color', 'r');
543         text(250, 600, sprintf('k = %i\n u = %i', k, u), 'color', 'r');
544         end
545     end
546
547
548     %precision and recall flag
549     pnr_flag = 0;
550
551     %precision and recall
552     T = struct2cell(frente);
553     T = T(1,:);
554     TF0 = contains(T,'a_0');
555     TF2 = contains(T,'a_2_frente_config_3');
556     TF21 = contains(T,'a_2_frente_config_4');
557     TF22 = contains(T,'a_2_frente_config_5');
558     TF4 = contains(T,'a_4_frente_config_3');
559     TF41 = contains(T,'a_4_frente_config_4');
560     TF42 = contains(T,'a_4_frente_config_5');
561
562     %setup
563     TN = 0;
564     FN = 0;
565     TP = 0;
566     FP = 0;
567
568     if debug_flag == 1
569     if TF0(k) == 1
570         %draw rectangle around real defect
571         x_defect = 1030;
```

```
572         y_defect = 541;
573         defect = [x_defect y_defect];
574         d_size = [20 20];
575 
576         %draw rectangle around defect
577         rx = [x_defect-d_size(1) x_defect+d_size(1) x_defect+d_size(1) ...
                x_defect-d_size(1) x_defect-d_size(1)];
578         ry = [y_defect-d_size(2) y_defect-d_size(2) y_defect+d_size(2) ...
                y_defect+d_size(2) y_defect-d_size(2)];
579 
580         plot(rx,ry,'r','linewidth',1)
581 
582         precision = TP/(TP+FP);
583         recall = TP/(TP+FN);
584         fnr = 1-recall;
585         fpr = FP/(FP+TN);
586 
587         if pnr_flag == 1
588         text(250, 500, sprintf('precision = %f\nrecall = %f\nfnr = ...
                %f\nfpr = %f', precision, recall, fnr, fpr), 'color', 'r');
589         end
590 
591     elseif TF4(k) == 1
592         %draw rectangle around real defect
593         x_defect = [1033; 1196; 1192];
594         y_defect = [119; 738; 1407];
595         d_size = [50 20; 20 20; 20 20];
596 
597         for d1 = 1:numel(x_defect)
598                 %draw rectangle around defect
599                 rx = [x_defect(d1)-d_size(d1,1) ...
                        x_defect(d1)+d_size(d1,1) x_defect(d1)+d_size(d1,1) ...
                        x_defect(d1)-d_size(d1,1) x_defect(d1)-d_size(d1,1)];
600                 ry = [y_defect(d1)-d_size(d1,2) ...
                        y_defect(d1)-d_size(d1,2) y_defect(d1)+d_size(d1,2) ...
                        y_defect(d1)+d_size(d1,2) y_defect(d1)-d_size(d1,2)];
601 
602                 plot(rx,ry,'r','linewidth',1)
603 
604                 if pnr_flag == 1
605                 text(250, 300 + d1*50, sprintf('relevant_outliers = ...
                        %f', relevant_outliersN), 'color', 'r');
606 
607                 text(250, 500 + d1*50, sprintf('precision = %f, recall ...
                        = %f', precision, recall), 'color', 'r');
608                 end
609         end
610 
611     elseif TF41(k) == 1
```

```
612         %draw rectangle around real defect
613         x_defect = [1043; 1187; 1174];
614         y_defect = [89; 706; 1390];
615         d_size = [50 20; 20 20; 20 20];
616 
617         for d1 = 1:numel(x_defect)
618                 %draw rectangle around defect
619                 rx = [x_defect(d1)-d_size(d1,1) ...
                        x_defect(d1)+d_size(d1,1) x_defect(d1)+d_size(d1,1) ...
                        x_defect(d1)-d_size(d1,1) x_defect(d1)-d_size(d1,1)];
620                 ry = [y_defect(d1)-d_size(d1,2) ...
                        y_defect(d1)-d_size(d1,2) y_defect(d1)+d_size(d1,2) ...
                        y_defect(d1)+d_size(d1,2) y_defect(d1)-d_size(d1,2)];
621 
622                 plot(rx,ry,'r','linewidth',1)
623 
624                 if pnr_flag == 1
625                 text(250, 300 + d1*50, sprintf('relevant_outliers = ...
                        %f', relevant_outliersN), 'color', 'r');
626 
627                 text(250, 500 + d1*50, sprintf('precision = %f, recall ...
                        = %f', precision, recall), 'color', 'r');
628                 end
629         end
630     elseif TF42(k) == 1
631         %draw rectangle around real defect
632         x_defect = [1036; 1177];
633         y_defect = [101; 728];
634         d_size = [50 20; 20 20];
635 
636         for d1 = 1:numel(x_defect)
637                 %draw rectangle around defect
638                 rx = [x_defect(d1)-d_size(d1,1) ...
                        x_defect(d1)+d_size(d1,1) x_defect(d1)+d_size(d1,1) ...
                        x_defect(d1)-d_size(d1,1) x_defect(d1)-d_size(d1,1)];
639                 ry = [y_defect(d1)-d_size(d1,2) ...
                        y_defect(d1)-d_size(d1,2) y_defect(d1)+d_size(d1,2) ...
                        y_defect(d1)+d_size(d1,2) y_defect(d1)-d_size(d1,2)];
640 
641                 plot(rx,ry,'r','linewidth',1)
642 
643                 if pnr_flag == 1
644                 text(250, 300 + d1*50, sprintf('relevant_outliers = ...
                        %f', relevant_outliersN), 'color', 'r');
645 
646                 text(250, 500 + d1*50, sprintf('precision = %f, recall ...
                        = %f', precision, recall), 'color', 'r');
647                 end
648         end
```

```
649
650     else
651         %precision and recall routine
652         relevant_outliersN = 0;
653         relevant_elements = 0;
654
655         positives = outliersN;
656         true_positives = relevant_outliersN;
657
658         precision = true_positives/positives;
659         recall = true_positives/relevant_elements;
660
661         if pnr_flag == 1
662             text(250, 350, sprintf('relevant_outliers = %f', ...
                    relevant_outliersN), 'color', 'r');
663             text(250, 500, sprintf('precision = %f, recall = %f', ...
                    precision, recall), 'color', 'r');
664         end
665     end
666     end
667
668
669 else
670     %do side config
671     %w8in revision - pointsCopy2 = line_fit(pointsCopy2,3, sampleSize, ...
            maxDistance, 1, [x y], 1);
672     %w8in revision - pointsCopy2 = line_fit(pointsCopy2,3, sampleSize, ...
            maxDistance, 0, [x y], 1);
673     %w8in revision - pointsCopy2 = poly_fit(pointsCopy2,1,1);
674     %w8in revision - pointsCopy2 = poly_fit(pointsCopy2,1,0);
675     %w8in revision - pointsCopy2 = poly_fit(pointsCopy2,1,1);
676
677     %w8in revision - outliersN = numel(pointsCopy2(:,1));
678     %w8in revision - plot(pointsCopy2(:,1),pointsCopy2(:,2),'yo')
679 end
680
681 %legend('edge points','inliers','line','corners','outliers');
682
683 if debug_flag == 1
684 hold off;
685
686 %save picture
687 %[height, width] = size(I);
688 %set(gcf,'position',[0 0 width height]);     % set size
689 export_fig(...
690     fullfile(['output/',folder(k).name],S{k}(u).name),...
691     '-png','-r500','-native');
692 end
693
```

```
694 fprintf('%s\n',S{k}(u).name);
695
696 if histo_flag == 1
697 %triangle plot
698 figure(f2);
699 subplot(3,2,1),
700 hold on;
701 for i=1:wu
702     histogram(distout{k,u}{i},...
703         'binedges',dedge,'normalization','probability'),...
704         title('line distances');
705 end
706 subplot(3,2,2),
707 hold on;
708 for i=1:wu
709     histogram(abs(orient_line{k,u}{i}),...
710         'binedges',oedge,'normalization','probability'),...
711         title('line orientations');
712 end
713 subplot(3,2,3), histogram(circle_distout{k,u,idc(1)},...
714     'binedges',cedge,'normalization','probability'),...
715     title('corner 1 distances');
716 subplot(3,2,4), histogram(abs(orient_corner{k,u,idc(1)}),...
717     'binedges',oedge,'normalization','probability'),...
718     title('corner 1 orientations');
719 subplot(3,2,5), histogram(circle_distout{k,u,idc(2)},...
720     'binedges',cedge,'normalization','probability'),...
721     title('corner 2 distances');
722 subplot(3,2,6), histogram(abs(orient_corner{k,u,idc(2)}),...
723     'binedges',oedge,'normalization','probability'),...
724     title('corner 2 orientations');
725
726 hold off;
727
728
729 %statistics on points distance to model
730 %if k == 1
731 %   %plot(str2double(S{k}(u).name(1:3)),mse{k,u},'g^')
732 %   plot(str2double(S{k}(u).name(1:3)),outliersN,'g^')
733 %   labels = cellstr(num2str(k));
734 %   %text(str2double(S{k}(u).name(1:3)),mse{k,u}, labels, ...
        'VerticalAlignment','bottom', ...
        'HorizontalAlignment','right','color','red')
735 %   text(str2double(S{k}(u).name(1:3)),outliersN, labels, ...
        'VerticalAlignment','bottom', ...
        'HorizontalAlignment','right','color','red')
736 %   if u == 1
737 %       hold on;
738 %   end
```

```
%else
%   ...
    %plot(str2double(S{k}(u).name(1:3)),mse{k,u},'x','color',colorstring(k))
%   ...
    plot(str2double(S{k}(u).name(1:3)),outliersN,'x','color',colorstring(k))
%   labels = cellstr(num2str(k));
%   %text(str2double(S{k}(u).name(1:3)),mse{k,u}, labels, ...
    'VerticalAlignment','bottom', ...
    'HorizontalAlignment','right','color','red')
%   text(str2double(S{k}(u).name(1:3)),outliersN, labels, ...
    'VerticalAlignment','bottom', ...
    'HorizontalAlignment','right','color','red')
%end

%save plot
%[height, width] = size(I);
%set(gcf,'position',[0 0 width height]);     % set size

%export_fig(...
%   fullfile(['output_samp/',folder(k).name],S{k}(u).name),...
%   '-png','-r96');
%fprintf('plot\n');
end

if u == 2
    %pause(inf);
end

%clean error count
err = 0;
%break;

catch e %e is an MException struct
    if err == 3
    if debug_flag == 1
    figure(f1);
    imshow(I);
    hold off;
    end

    if contains(folder(k).name,'config_9') == 1 || ...
        contains(folder(k).name,'config_6') == 1 || ...
        contains(folder(k).name,'config_10') == 1
        op1 = nan(1,16);
        op2 = nan(1,4);
    else
        op1 = nan(1,5);
        op2 = nan(1,2);
    end
```

```
780
781     %metrics on error
782     mse{k,u} = op1;
783     harris{k,u} = op1;
784     mse_orient_line{k,u} = op1;
785     line_sign{k,u} = op1;
786     line_outliers{k,u} = op1;
787     line_mean{k,u} = op1;
788     line_std{k,u} = op1;
789
790     circle_radius_relevant{k,u} = op2;
791     corner_sign_relevant{k,u} = op2;
792     circle_distout_mean{k,u} = op2;
793     circle_distout_std{k,u} = op2;
794     circle_outliers_relevant{k,u} = op2;
795
796     pdist_mean{k,u} = op2;
797     pdist_std{k,u} = op2;
798     para_outliers_relevant{k,u} = op2;
799
800     t{k,u} = nan(1,17);
801
802     %label
803     if debug_flag == 1
804     text(250, 250, 'ERROR', 'color', 'r');
805     export_fig(...
806     fullfile(['output/',folder(k).name],S{k}(u).name),...
807         '-png','-r96','-native');
808     end
809
810     fprintf('%s\n',S{k}(u).name);
811     fprintf(2,'-----\nThe identifier was:\n%s',e.identifier);
812     fprintf(2,'There was an error! The message was:\n%s\n',e.message);
813     fprintf(2,'Line %i\n',e.stack.line);
814     fprintf(2,'Files %s\n',e.stack.name);
815     % more error handling...
816     fprintf('ERROR WITH %s\n-----\n',S{k}(u).name);
817
818     %write error log
819     fid = fopen('error_log','a+');
820     fprintf(fid,'%s\n',e.message);
821     for i=1:length(e.stack)
822         fprintf(fid,'%s at %i\n',e.stack(i).name,e.stack(i).line);
823     end
824
825     fclose(fid);
826
827     %break;
828     err = 0;
```

```
829     else
830
831         err = err +1;
832
833         %CC = bwconncomp(BWoutline);
834         %if CC.NumObjects > 1
835         %   break;
836         %   %err = err -2;
837         %else
838         %   err = err +1;
839         %end
840     end
841 end %end try-catch
842 end %while
843
844 t{k,u}(17) = toc(tstart);
845
846 end %u-loop
847
848     A = struct2cell(S{k});
849     A1 = A(1,:); %names of pictures
850
851     Ac(1:1000,1) = string(folder(k).name);
852     %Ac = Ac';
853
854     %recomp names for configs with all sides
855     if contains(folder(k).name,'config_9') == 1 || ...
           contains(folder(k).name,'config_6') == 1 || ...
           contains(folder(k).name,'config_10') == 1
856         A2 = repmat(A1,16,1);
857         A3 = repmat(A1,4,1);
858     else
859         A2 = repmat(A1,5,1);
860         A3 = repmat(A1,2,1);
861     end
862
863     A2 = A2(:);
864     A3 = A3(:);
865
866     nu = numel(A1); %images
867     ni = numel(A2); %line
868     ny = numel(A3); %corner
869     nl = numel(cell2mat(mse(k,:))');
870     nlc = numel(cell2mat(circle_radius_relevant(k,:))');
871
872     if contains(folder(k).name,'0_frente_config_3') == 1
873         defect_l = [zeros(5,1); repmat([0 1 0 0 0]',nu-1,1)];
874         defect_c = zeros(ny,1);
875
```

```
876     elseif contains(folder(k).name,'2_frente_config_3') == 1 || ...
            contains(folder(k).name,'2_frente_config_5') == 1
877         %all zeros
878         defect_l = repmat([0 0 0 0 0]',nu,1);
879         defect_c = zeros(ny,1);
880
881     elseif contains(folder(k).name,'2_frente_config_5') == 1
882         defect_l = repmat([0 0 0 0 0]',nu,1);
883         defect_c = zeros(ny,1);
884
885     elseif contains(folder(k).name,'2_frente_config_6') == 1
886         defect_l = repmat(zeros(16,1),nu,1);
887         defect_c = zeros(ny,1);
888
889     %needs revision
890     elseif contains(folder(k).name,'2_frente_config_9') == 1
891         defect_l = repmat(zeros(16,1),nu,1);
892         defect_c = zeros(ny,1);
893
894     %upper left corner, undetected
895     elseif contains(folder(k).name,'2_frente_config_10') == 1
896         defect_l = repmat(zeros(16,1),nu,1);
897         defect_c = repmat([0 0]',nu*2,1);
898
899     elseif contains(folder(k).name,'4_frente_config_3') == 1
900         defect_l = [repmat([0 0 0 0 0]',12,1); repmat([0 0 1 0 ...
                1]',nu-12,1)];
901         defect_c = [repmat([1 0]',12,1); zeros(ny-24,1)];
902
903     %does not work well
904     elseif contains(folder(k).name,'4_frente_config_4') == 1
905         defect_l = repmat([0 0 0 0 0]',nu,1);
906         defect_c = repmat([0 0]',nu,1);
907
908     %same
909     elseif contains(folder(k).name,'4_frente_config_5') == 1
910         defect_l = repmat([0 0 0 0 0]',nu,1);
911         defect_c = repmat([0 0]',nu,1);
912
913     %needs revision, bad skeleton obtained
914     elseif contains(folder(k).name,'4_frente_config_9') == 1
915         defect_l = [repmat([0 0 0 0 0]',nu,1); repmat([0 0 1 0 ...
                1]',nu,1); repmat([0 0 0]',nu*2,1)];
916         defect_c = repmat([1 0 0 0]',nu,1);
917
918     %needs revision
919     elseif contains(folder(k).name,'4_frente_config_10') == 1
920         defect_l = [repmat([0 0 0 0 0]',nu,1); repmat([0 0 1 0 ...
                1]',nu,1); repmat([0 0 0]',nu*2,1)];
```

```
921         defect_c = repmat([1 0 0 1]',nu,1);
922     else
923         defect_l = zeros(ni,1);
924         defect_c = zeros(ny,1);
925     end
926
927     %line performance parameters
928     par_mse = pnr(cell2mat(mse(k,:))', defect_l, [0 0.6])';
929     par_mse_ori_l = pnr(cell2mat(mse_orient_line(k,:))', defect_l, [0 ...
            50])';
930     par_lmean = pnr(cell2mat(line_mean(k,:))', defect_l, [0 1])';
931     par_lstd = pnr(cell2mat(line_std(k,:))', defect_l, [0 0.5])';
932     par_lsign = pnr(cell2mat(line_sign(k,:))', defect_l, [0 0.5])';
933     par_lout = pnr(cell2mat(line_outliers(k,:))', defect_l, [0 10])';
934     par_harris = pnr(cell2mat(harris(k,:))', defect_l, [0 0.5])';
935
936     %corner performance parameters
937     par_cr = pnr(cell2mat(circle_radius_relevant(k,:))', defect_c, [14 ...
            20])';
938     par_cs = pnr(cell2mat(corner_sign_relevant(k,:))', defect_c, [2 4])';
939     par_cdmean = pnr(cell2mat(circle_distout_mean(k,:))', defect_c, [0 ...
            1.3])';
940     par_cdstd = pnr(cell2mat(circle_distout_std(k,:))', defect_c, [0 1])';
941     par_co = pnr(cell2mat(circle_outliers_relevant(k,:))', defect_c, [0 ...
            3])';
942     par_pdmean = pnr(cell2mat(pdist_mean(k,:))', defect_c, [0 0.8])';
943     par_pdstd = pnr(cell2mat(pdist_std(k,:))', defect_c, [0 0.75])';
944     par_po = pnr(cell2mat(para_outliers_relevant(k,:))', defect_c, [0 1])';
945
946     par_metric_l = ...
            ['par_mse','par_mse_ori_l','par_lmean','par_lstd','par_lsign','par_lout','par_harris'];
947     par_metric_c = ...
            ['par_cr','par_cs','par_cdmean','par_cdstd','par_co','par_pdmean','par_pdstd','par_po'];
948
949     TL{k} = table(Ac(1:nl), A2(1:nl), ...
            logical(defect_l(1:nl)),cell2mat(mse(k,:))', ...
            cell2mat(mse_orient_line(k,:))', cell2mat(line_mean(k,:))', ...
            cell2mat(line_std(k,:))', cell2mat(line_sign(k,:))', ...
            cell2mat(line_outliers(k,:))',cell2mat(harris(k,:))', ...
950         'variablenames',...
951         {'config','files','defect','line_mse','ori_mse','line_mean','line_std', ...
                'line_sign','line_outliers','harris'});
952     TC{k} = table(Ac(1:nlc), A3(1:nlc),logical(defect_c(1:nlc)),...
953         cell2mat(circle_radius_relevant(k,:))', ...
                cell2mat(corner_sign_relevant(k,:))', ...
                cell2mat(circle_distout_mean(k,:))', ...
                cell2mat(circle_distout_std(k,:))', ...
                cell2mat(circle_outliers_relevant(k,:))', ...
```

```
            cell2mat(pdist_mean(k,:))', cell2mat(pdist_std(k,:))', ...
            cell2mat(para_outliers_relevant(k,:))', ...
954         'variablenames',...
955         {'config','files','defect','circle_radius','corner_sign','cd_mean',...
956             'cd_std','circle_outliers','pd_mean','pd_std','pd_out'});
957     TM{k} = table(par_mse, par_mse_ori_l, par_lmean, par_lstd, ...
            par_lsign, par_lout,  par_harris, par_cr, par_cs, par_cdmean, ...
            par_cdstd, par_co, par_pdmean, par_pdstd, par_po, 'rownames', ...
            {'acc','bacc','ppv','rec','tnr'});
958
959     if k == 1
960         ju = 1;
961     end
962
963     %write tables
964     writetable(TL{k},'tables_v1.xlsx','sheet',ju);
965     writetable(TC{k},'tables_v1.xlsx','sheet',ju+1);
966
967     ju=ju+2;
968
969     if plot_flag == 1
970     figure(f2);
971     %varied plots
972     do_plot(mse,k,defect_l,folder(k).name,0,0.15);
973     do_plot(mse_orient_line,k,defect_l,folder(k).name,0);
974     do_plot(line_mean,k,defect_l,folder(k).name,0);
975     do_plot(line_std,k,defect_l,folder(k).name,0);
976     do_plot(line_sign,k,defect_l,folder(k).name,0);
977     do_plot(line_outliers,k,defect_l,folder(k).name,0);
978     do_plot(harris,k,defect_l,folder(k).name,0);
979
980     %weights = [1 1 0 0 0 0];
981
982     %w_value{k,:} = ...
            weights(1)*(cell2mat(mse(k,:))'/max(cell2mat(mse(k,:)))) + ...
983     %           ...
            weights(2)*(cell2mat(mse_orient_line(k,:))'/max(cell2mat(mse_orient_line(k,:))))
            + ...
984     %           weights(3)*cell2mat(line_mean(k,:))' + ...
985     %           weights(4)*cell2mat(line_std(k,:))' + ...
986     %           weights(5)*cell2mat(line_sign(k,:))' + ...
987     %           weights(6)*cell2mat(line_outliers(k,:))';
988
989     %do_plot(w_value,defect_l,k,0);
990
991     do_plot(circle_radius_relevant,k,defect_c,folder(k).name,1);
992     do_plot(corner_sign_relevant,k,defect_c,folder(k).name,1);
993     do_plot(circle_distout_mean,k,defect_c,folder(k).name,1);
994     do_plot(circle_distout_std,k,defect_c,folder(k).name,1);
```

```
995      do_plot(circle_outliers_relevant,k,defect_c,folder(k).name,1);
996      do_plot(pdist_mean,k,defect_c,folder(k).name,1);
997      do_plot(pdist_std,k,defect_c,folder(k).name,1);
998      do_plot(para_outliers_relevant,k,defect_c,folder(k).name,1);
999      end
1000 end %end of k
1001
1002 table_line = vertcat(TL{1:k});
1003 table_corner = vertcat(TC{1:k});
1004
1005 %tree_line = fitctree(C(:,3:8),C.defect);
1006
1007 save v1;
```

```
1  function [BWoutline, BWper] = segmentation(I, fudgeFactor, area, debug) ...
       %, folder, name)
2
3      % testing edge detection (canny performs best, sobel accentuates ...
           defects)
4      [¬, threshold] = edge(I, 'canny');
5      %fudgeFactor = 0.5;
6      BWs = edge(I,'canny', threshold * fudgeFactor);
7
8      se90 = strel('line', 3, 90);
9      se0 = strel('line', 3, 0);
10
11     % dilate image
12     BWsdil = imdilate(BWs, [se90 se0]);
13
14     % fill interior holes
15     BWdfill = imfill(BWsdil, 'holes');
16
17     % remove connected objects on border
18     %BWnobord = imclearborder(BWdfill, 4);
19     %
20     % smooth the image (needs revision)
21     seD = strel('diamond',1);
22     BWfinal = imerode(BWdfill,seD);
23     BWfinal = imerode(BWfinal,seD);
24
25     % filter areas and perimeter
26     %BWao = bwareaopen(BWfinal,area);
27     BWao = bwpropfilt(BWfinal,'area',[area 260000]);
28     BWper = bwpropfilt(BWao,'perimeter',[2800 3200]);
29
30     % create perimeter
31     BWoutline = bwperim(BWper);
32     %Segout = I;
```

```
33      %Segout(BWoutline) = 255;
34
35      if debug == 1
36      subplot(2,3,1), imshow(I), title('original');
37      subplot(2,3,2), imshow(BWs), title('binary gradient mask');
38      subplot(2,3,3), imshow(BWsdil), title('dilated gradient mask');
39      subplot(2,3,4), imshow(BWdfill), title('binary image with filled ...
            holes');
40      %subplot(3,3,5), imshow(BWnobord), title('cleared border image');
41      subplot(2,3,5), imshow(BWfinal), title('segmented image');
42      subplot(2,3,6), imshow(BWper), title('remove small areas');
43      %subplot(3,3,7), imshow(BWoutline), title('outline');
44      end
45
46      %dilated version of BWao (discontinued)
47      %BWaodil = imdilate(BWao, strel('diamond',10));
48
49      %save picture
50      %[height, width] = size(I);
51      %set(gcf,'position',[0 0 1920 1080]);      % set size
52      %export_fig(fullfile(['output_seg/',folder], name),'-png','-r96');
53      %fprintf('%s\n',name);
54
55      %hold off
56
57      % output image for manipulation
58      %imtool(Segout);
59
60      %spy(BWoutline);
61
62      %% get x and y derivatives using sobel by default
63      %[Gx,Gy] = imgradientxy(I);
64      %Gmagx = imgradient(Gx);
65
66      % pixel connectivity (leave largest obj)
67      %CC = bwconncomp(BWline);
68      %while CC.NumObjects > 1
69      %   numPixels = cellfun(@numel,CC.PixelIdxList);
70      %   [smallest, idx] = min(numPixels);
71      %   BWline(CC.PixelIdxList{idx}) = 0;
72      %   CC = bwconncomp(BWline);
73      %end
74
75      %subplot(3,3,9)
76  end
```

```
1   function [BWoutline, BWper] = segmentation9(I, fudgeFactor, area, ...
        debug) %, folder, name)
```

```
2
3       % testing edge detection (canny performs best, sobel accentuates ...
            defects)
4       [¬, threshold] = edge(I, 'canny');
5       %fudgeFactor = 0.5;
6       BWs = edge(I,'canny', threshold * fudgeFactor);
7
8       se90 = strel('line', 3, 90);
9       se0 = strel('line', 3, 0);
10
11      % dilate image
12      BWsdil = imdilate(BWs, [se90 se0]);
13
14      % fill interior holes
15      BWdfill = imfill(BWsdil, 'holes');
16
17      % remove connected objects on border
18      %BWnobord = imclearborder(BWdfill, 4);
19      %
20      % smooth the image (needs revision)
21      seD = strel('diamond',1);
22      BWfinal = imerode(BWdfill,seD);
23      BWfinal = imerode(BWfinal,seD);
24
25      % filter areas and perimeter
26      %BWao = bwareaopen(BWfinal,area);
27      BWao = bwpropfilt(BWfinal,'area',[area 260000]);
28      BWper = bwpropfilt(BWao,'perimeter',[2800 3200]);
29
30      % create perimeter
31      BWoutline = bwperim(BWper);
32      %Segout = I;
33      %Segout(BWoutline) = 255;
34
35      BWoutlinedil = imdilate(BWoutline, [se90 se0]);
36
37      BWs = BWs - BWoutlinedil;
38
39      BWs = imbinarize(BWs);
40
41      % dilate image
42      BWsdil = imdilate(BWs, [se90 se0]);
43
44      % fill interior holes
45      BWdfill = imfill(BWsdil, 'holes');
46
47      % smooth the image (needs revision)
48      BWfinal = imerode(BWdfill,seD);
49      BWfinal = imerode(BWfinal,seD);
```

```
50
51      % filter areas and perimeter
52      %BWao = bwareaopen(BWfinal,area);
53      BWao = bwpropfilt(BWfinal,'area',[area 260000]);
54      BWper = bwpropfilt(BWao,'perimeter',[2800 3200]);
55
56      % create perimeter
57      BWoutline = bwperim(BWper);
58      %Segout = I;
59      %Segout(BWoutline) = 255;
60
61      if debug == 1
62      subplot(2,3,1), imshow(I), title('original');
63      subplot(2,3,2), imshow(BWs), title('binary gradient mask');
64      subplot(2,3,3), imshow(BWsdil), title('dilated gradient mask');
65      subplot(2,3,4), imshow(BWdfill), title('binary image with filled ...
            holes');
66      %subplot(3,3,5), imshow(BWnobord), title('cleared border image');
67      subplot(2,3,5), imshow(BWfinal), title('segmented image');
68      subplot(2,3,6), imshow(BWper), title('remove small areas');
69      %subplot(3,3,7), imshow(BWoutline), title('outline');
70      end
71
72      %dilated version of BWao (discontinued)
73      %BWaodil = imdilate(BWao, strel('diamond',10));
74
75      %save picture
76      %[height, width] = size(I);
77      %set(gcf,'position',[0 0 1920 1080]);     % set size
78      %export_fig(fullfile(['output_seg/',folder], name),'-png','-r96');
79      %fprintf('%s\n',name);
80
81      %hold off
82
83      % output image for manipulation
84      %imtool(Segout);
85
86      %spy(BWoutline);
87
88      %% get x and y derivatives using sobel by default
89      %[Gx,Gy] = imgradientxy(I);
90      %Gmagx = imgradient(Gx);
91
92      % pixel connectivity (leave largest obj)
93      %CC = bwconncomp(BWline);
94      %while CC.NumObjects > 1
95      %   numPixels = cellfun(@numel,CC.PixelIdxList);
96      %   [smallest, idx] = min(numPixels);
97      %   BWline(CC.PixelIdxList{idx}) = 0;
```

```
    %   CC = bwconncomp(BWline);
    %end

    %subplot(3,3,9)
end
```

```
function [points, l_out, brightIdx] = line_fit(points, i, sampleSize, ...
    maxDistance, vertical, doPlot, bwim)
max_bright = 0;
l_out = cell(1,i);
for k=1:i
    if vertical == 1
        % ransac on vertical line
        % switched x and y values for distance evaluation to avoid ...
            vertical line fitting
        fitLineFcn = @(points) polyfit(points(:,2),points(:,1),1); % ...
            fit function using polyfit
        evalLineFcn = ...   % distance evaluation function
          @(model, points) sum((points(:,1) - polyval(model, ...
              points(:,2))).^2,2);

        [¬, inlierIdx] = ransac(points,fitLineFcn,evalLineFcn, ...
          sampleSize,maxDistance);

        modelInliers = polyfit(points(inlierIdx,1),points(inlierIdx,2),1);

        inlierPts = points(inlierIdx,:);

        if doPlot > 0
            %plot inliers and lines
            plot(inlierPts(:,1),inlierPts(:,2),'g.')
            %plot(inlierPts(:,1),inlierPts(:,2),'go')
            %plot(points(¬inlierIdx,1),points(¬inlierIdx,2),'mo')
        end

        %switched x and y
        y_min = min(inlierPts(:,2));
        y_max = max(inlierPts(:,2));
        y1 = [y_min; y_max];
        x1 = (y1 - modelInliers(2))/modelInliers(1);
        if doPlot == 1
            plot(x1, y1,'r-','linewidth',1)
        elseif doPlot == 2
            plot(x1, y1, 'y-')
        elseif doPlot == 3
            plot(x1, y1, 'b-','linewidth',1)
        end

```

```
39          l_out{k} = [x1 y1];
40
41      else
42          % ransac on horizontal line
43          fitLineFcn = @(points) polyfit(points(:,1),points(:,2),1); % ...
                fit function using polyfit
44          evalLineFcn = ...   % distance evaluation function
45            @(model, points) sum((points(:,2) - polyval(model, ...
                  points(:,1))).^2,2);
46
47          [¬, inlierIdx] = ransac(points,fitLineFcn,evalLineFcn, ...
48            sampleSize,maxDistance);
49
50          modelInliers = polyfit(points(inlierIdx,1),points(inlierIdx,2),1);
51
52          inlierPts = points(inlierIdx,:);
53
54          if doPlot > 0
55              %plot inliers and lines
56              plot(inlierPts(:,1),inlierPts(:,2),'g.')
57          end
58          x_min = min(inlierPts(:,1));
59          x_max = max(inlierPts(:,1));
60          x1 = [x_min; x_max];
61          y1 = modelInliers(1)*x1 + modelInliers(2);
62          if doPlot > 0
63              plot(x1, y1, 'r-')
64          end
65          l_out{k} = [x1 y1];
66      end
67
68      if nargin == 7
69          [y, x] = find(bwim);
70          bwPts = [x y];
71          brightPts = intersect(inlierPts, bwPts,'rows');
72          brightness = numel(brightPts(:,1))/numel(inlierPts(:,1));
73          if brightness > max_bright
74              max_bright = brightness;
75              brightIdx = k;
76          end
77      end
78
79      %cleanup
80      points(inlierIdx,:) = [];
81  end
82  end
```

```
function [points, inlierPts, P] = poly_fit(points,ii, maxDistance, ...
    vertical, debug)
for k=1:ii
    if vertical == 1
        N = 2;           % second-degree polynomial
        %maxDistance = 3; % maximum allowed distance for a point to be ...
            inlier

        vPts(:,1) = points(:,2);
        vPts(:,2) = points(:,1);

        [P, inlierIdx] = fitPolynomialRANSAC(vPts,N,maxDistance);
        yRecoveredCurve = polyval(P,vPts(:,1));

        % -- plot(vPts(:,1),yRecoveredCurve,'r-','LineWidth',1)
        % -- plot(vPts(inlierIdx, 1),vPts(inlierIdx, 2),'g.')

        if debug == 1
        plot(yRecoveredCurve,vPts(:,1),'r-','LineWidth',1)
        %plot(vPts(inlierIdx, 2),vPts(inlierIdx, 1),'g.')
        end

        inlierPts = points(inlierIdx,:);
        points(inlierIdx,:) = [];
    else
        N = 2;           % second-degree polynomial
        %maxDistance = 3; % maximum allowed distance for a point to be ...
            inlier

        [P, inlierIdx] = fitPolynomialRANSAC(points,N,maxDistance);
        yRecoveredCurve = polyval(P,points(:,1));

        if debug == 1
        plot(points(:,1),yRecoveredCurve,'r-','LineWidth',1)
        %plot(points(inlierIdx, 1),points(inlierIdx, 2),'g.')
        end

        inlierPts = points(inlierIdx,:);
        points(inlierIdx,:) = [];
    end
end
```

```
function [dist] = dist_circle(points,par)
    %points --> vector containing the data [x y]
    %par --> parameters of the desired circle [a b R]
    %dist will be a vector with same number of rows as 'points'

```

```
6       dist = ...
            abs(sqrt((points(:,1)-par(1)).^2+(points(:,2)-par(2)).^2)-par(3));
7   end
```

```
1   function [dist] = dist_outliers(p1,p2,points)
2   dist = zeros(numel(points(:,1),1));
3       for k=1:numel(points(:,1))
4           p0 = points(k,:);
5           %dist = ...
                abs((y2-y1)*x0-(x2-x1)*y0+x2*y1-y2*x1)/sqrt((y2-y1)^2+(x2-x1)^2);
6           dist(k,1) = abs((p2(2)-p1(2))*p0(1)-...
7               (p2(1)-p1(1))*p0(2)+p2(1)*p1(2)-p2(2)*p1(1))...
8               /sqrt((p2(2)-p1(2))^2+(p2(1)-p1(1))^2);
9       end
10  end
```

```
1   function [distance] = dist_para_dy(points,par)
2       %points [x y]
3       %par [a b c]
4       syms f(x)
5       dy = zeros(1, numel(points(:,1)));
6       f(x) = par(1)*x^2 + par(2)*x +par(3);
7
8       for i=1:numel(points(:,1))
9       x1 = points(i,1);
10      y1 = points(i,2);
11
12      dy(i) = f(x1) - y1;
13      end
14      distance = abs(dy)';
15  end
```

```
1   function [out_sign] = sign_thresh(A,threshold)
2   S = diff(sign(A),1);
3   id = find(abs(S) == 2);
4   id = [0 id];
5   n = numel(id);
6   m = numel(A);
7   B = cell(n,1);
8   for j = 1:n
9       if  j == n
10          B{j} = A(id(j)+1:m);
11      else
12          B{j} = A(id(j)+1:id(j+1));
13      end
```

```
14
15      %apply threshold
16      if numel(B{j}) ≤ threshold
17          B{j} = [];
18      end
19  end
20
21  C = cell2mat(B');
22
23  out_sign = numel(find(abs(diff(sign(C),1)) == 2));
```

```
1   function [relevantPts_out, relevant_outliers, distout, mse, line_rect, ...
        idc, harris] = rect(points, line, vertical,corners,I,bwoutline, debug)
2       %filter points throught rectangle based on line
3       dist = 25; %width of the square will be 2*dist
4
5       if vertical == 1
6
7           %5 zones
8           zo = 5;
9
10          %draw relevant rectangle
11          idc = dsearchn(corners,line);
12
13          relevant_corners = corners(idc,:);
14
15          [¬, i] = min(relevant_corners(:,2));
16
17          %draw rectangle around the relevant line
18          rx_line = [relevant_corners(i,1)-dist ...
                relevant_corners(i,1)+dist relevant_corners(3-i,1)+dist ...
                relevant_corners(3-i,1)-dist relevant_corners(i,1)-dist];
19          ry_line = [relevant_corners(i,2)+dist ...
                relevant_corners(i,2)+dist relevant_corners(3-i,2)-dist ...
                relevant_corners(3-i,2)-dist relevant_corners(i,2)+dist];
20
21      else
22          zo = 3;
23
24          %draw relevant rectangle
25          idc = dsearchn(corners,line);
26
27          relevant_corners = corners(idc,:);
28
29          [¬, i] = min(relevant_corners(:,1));
30
31          %draw rectangle around the relevant line
```

```
32          rx_line = [relevant_corners(i,1)+dist ...
                relevant_corners(3-i,1)-dist relevant_corners(3-i,1)-dist ...
                relevant_corners(i,1)+dist relevant_corners(i,1)+dist];
33          ry_line = [relevant_corners(i,2)-dist ...
                relevant_corners(3-i,2)-dist relevant_corners(3-i,2)+dist ...
                relevant_corners(i,2)+dist relevant_corners(i,2)-dist];
34
35      end
36
37      %prelocate for speed
38      rx = cell(1,zo);
39      ry = cell(1,zo);
40
41      %plot relevant rectangle
42      %plot(rx_line,ry_line,'c','linewidth',1)
43
44      %export
45      line_rect = [rx_line(1) ry_line(1) 2*dist ...
            relevant_corners(3-i,2)-relevant_corners(i,2)+2*dist];
46
47      %relevant points
48      in = inpolygon(points(:,1),points(:,2),rx_line,ry_line);
49      relevantPts = points(in,:);
50
51      %redo lines with ransac
52      m = numel(relevantPts(:,1));
53      n = round(m/zo);
54
55      if vertical == 1
56      %5 sections
57      id(1,:) = [1 n];
58      id(2,:) = [n+1 2*n];
59      id(3,:) = [2*n+1 3*n];
60      id(4,:) = [3*n+1 4*n];
61      id(5,:) = [4*n+1 m];
62      else
63      %3 sections
64      id(1,:) = [1 n];
65      id(2,:) = [n+1 2*n];
66      id(3,:) = [2*n+1 m];
67      end
68
69      sampleSize = 5;
70      maxDistance = 1;
71
72      if vertical == 1
73      %order points by yy
74      [y, idy] = sort(relevantPts(:,2));
75      relevantPts = [relevantPts(idy,1) y];
```

```
76
77      %create 5 rectangles
78      for i=1:5
79      rx{i} = [relevantPts(id(i,1),1)-dist relevantPts(id(i,1),1)+dist ...
            relevantPts(id(i,2),1)+dist relevantPts(id(i,2),1)-dist ...
            relevantPts(id(i,1),1)-dist];
80      ry{i} = [relevantPts(id(i,1),2) relevantPts(id(i,1),2) ...
            relevantPts(id(i,2),2) relevantPts(id(i,2),2) ...
            relevantPts(id(i,1),2)];
81      end
82
83      else
84      %order points by xx
85      [x, idx] = sort(relevantPts(:,1));
86      relevantPts = [x relevantPts(idx,2)];
87
88      %create 3 rectangles
89      for i=1:3
90      rx{i} = [relevantPts(id(i,1),1) relevantPts(id(i,2),1) ...
            relevantPts(id(i,2),1) relevantPts(id(i,1),1) ...
            relevantPts(id(i,1),1)];
91      ry{i} = [relevantPts(id(i,1),2)-dist relevantPts(id(i,2),2)-dist ...
            relevantPts(id(i,2),2)+dist relevantPts(id(i,1),2)+dist ...
            relevantPts(id(i,1),2)-dist];
92      end
93      end
94
95      %prelocate for speed
96      harris = zeros(1,zo);
97
98      J = imadjust(I,[],[],0.5);
99      f = detectHarrisFeatures(J);
100
101     [y1,x1] = find(bwoutline);
102     %remove f.Location decimals for intersection
103     fo = intersect([x1 y1], fix(f.Location),'rows');
104
105     for i=1:zo
106         if debug == 1
107         plot(rx{i},ry{i},'c','linewidth',1)
108         end
109
110         %in = inpolygon(f.Location(:,1),f.Location(:,2), rx{i}, ry{i});
111         in = inpolygon(fo(:,1),fo(:,2), rx{i}, ry{i});
112         plot(f.Location(in,1),f.Location(in,2))
113         harris(i) = numel(find(in));
114     end
115
116     if vertical == 1
```

```
117         [relevant_outliers{1}, ¬, coef1] = poly_fit(relevantPts(1:n,:), ...
                1, maxDistance, 1, debug);
118         [relevant_outliers{2}, line1] = ...
                line_fit(relevantPts(n+1:2*n,:),1,sampleSize, ...
                maxDistance,1,debug);
119         [relevant_outliers{3}, line2] = ...
                line_fit(relevantPts(2*n+1:3*n,:),1,sampleSize, ...
                maxDistance,1,debug);
120         [relevant_outliers{4}, line3] = ...
                line_fit(relevantPts(3*n+1:4*n,:),1,sampleSize, ...
                maxDistance,1,debug);
121         [relevant_outliers{5}, ¬, coef2] = ...
                poly_fit(relevantPts(4*n+1:m,:), 1, maxDistance, 1, debug);
122     else
123         [relevant_outliers{1}, ¬, coef1] = poly_fit(relevantPts(1:n,:), ...
                1, maxDistance, 0, debug);
124         [relevant_outliers{2}, line1] = ...
                line_fit(relevantPts(n+1:2*n,:),1,sampleSize, ...
                maxDistance,0,debug);
125         [relevant_outliers{3}, ¬, coef2] = ...
                poly_fit(relevantPts(2*n+1:m,:), 1, maxDistance, 0, debug);
126     end
127
128     %plot debug for sorted points
129     %plot(relevantPts(1:n,1),relevantPts(1:n,2),'go')
130     %plot(relevantPts(n+1:m,1),relevantPts(n+1:m,2),'ko')
131
132     %%clean outliers
133     %in = inpolygon(outliers(:,1),outliers(:,2),rx_line,ry_line);
134     %outliers(in,:) = [];
135
136     %add line outliers to corner outliers for plotting
137     if vertical == 1
138     relevant_outliers_plot = [relevant_outliers{1}; ...
            relevant_outliers{2}; relevant_outliers{3}; ...
            relevant_outliers{4}; relevant_outliers{5}];
139     else
140     relevant_outliers_plot = [relevant_outliers{1}; ...
            relevant_outliers{2}; relevant_outliers{3}];
141     end
142
143     %plot outliers
144     if debug == 1
145     plot(relevant_outliers_plot(:,1),relevant_outliers_plot(:,2),'mo')
146     end
147
148     %distances
149     if vertical == 1
```

```
150     distout{1} = dist_para_dy([relevantPts(1:n,2) relevantPts(1:n,1)], ...
            coef1);
151     distout{2} = ...
            dist_outliers(line1{1}(1,:),line1{1}(2,:),relevantPts(n+1:2*n,:));
152     distout{3} = ...
            dist_outliers(line2{1}(1,:),line2{1}(2,:),relevantPts(2*n+1:3*n,:));
153     distout{4} = ...
            dist_outliers(line3{1}(1,:),line3{1}(2,:),relevantPts(3*n+1:4*n,:));
154     distout{5} = dist_para_dy([relevantPts(4*n+1:m,2) ...
            relevantPts(4*n+1:m,1)], coef2);
155     else
156     distout{1} = dist_para_dy([relevantPts(1:n,1) relevantPts(1:n,2)], ...
            coef1);
157     distout{2} = ...
            dist_outliers(line1{1}(1,:),line1{1}(2,:),relevantPts(n+1:2*n,:));
158     distout{3} = dist_para_dy([relevantPts(2*n+1:m,1) ...
            relevantPts(2*n+1:m,2)], coef2);
159     end
160
161     %%merge distances
162     %distout = [distout1; distout2; distout3];
163
164     %mean-squared error
165     mse1 = do_mse(distout{1});
166     mse2 = do_mse(distout{2});
167     mse3 = do_mse(distout{3});
168     if vertical == 1
169     mse4 = do_mse(distout{4});
170     mse5 = do_mse(distout{5});
171     end
172
173
174     if vertical == 1
175     mse = [mse1 mse2 mse3 mse4 mse5];
176     else
177     mse = [mse1 mse2 mse3];
178     end
179
180     if vertical == 1
181     relevantPts_out{1} = relevantPts(1:n,:);
182     relevantPts_out{2} = relevantPts(n+1:2*n,:);
183     relevantPts_out{3} = relevantPts(2*n+1:3*n,:);
184     relevantPts_out{4} = relevantPts(3*n+1:4*n,:);
185     relevantPts_out{5} = relevantPts(4*n+1:m,:);
186     else
187     relevantPts_out{1} = relevantPts(1:n,:);
188     relevantPts_out{2} = relevantPts(n+1:2*n,:);
189     relevantPts_out{3} = relevantPts(2*n+1:m,:);
190     end
```

```
function [Orientations] = skeletonOrientation(skel,blksz)
% SKELETONORIENTATION Calculate the local orientation of a skeleton
%
%Inputs:
%  skel:  MxN binary skeleton image.
%
%  blksz: Size of block to look around for local orientation
%         Both elements must be odd and greater than or equal to three
%         blksz can be a 1x2 vector meaning [row x col] block size or 1x1
%               for square block
%         optional, defaults to [5 5]
%
%Outputs:
%  Orientation: image that is the size of skel with zeros outside of skel
%               and orientations elsewhere
%
%


%
% Copyright 2013 The MathWorks, Inc.
% SCd 9/25/2013
%

    %Error checking:
    assert(nargin==1||nargin==2,'One or two inputs expected');
    assert(islogical(skel),'skel should be logical');
    assert(ismatrix(skel),'skel should be a matrix');
    sz = size(skel);
    assert(isequal(sz,size(skel)),'Sizes of M and C expected to be equal');

    %Handling blksz
    if nargin == 1
        %Default
        blksz = [5 5];
    else
        %assertions
        assert(all(blksz≥3),'blksz elements must be greater than or ...
            equal to three');
        assert(all(mod(blksz,2)==1),'blksz elements expected to be odd');
        if isscalar(blksz)
            blksz = blksz([1,1]);
        else
            assert(isequal(size(blksz),[1 2]),'blksz is expected to be ...
                a variable of size [1x1] or [1x2]');
        end
    end

    %Find the skeleton pixels' index
    [row,col] = find(skel);
```

```
48      npts      = numel(row);
49
50      %Pad the array and offset the rows/cols so every local block fits
51      padAmount = floor(blksz./2); %distance from center to edge of block
52      skelPad   = padarray(skel,padAmount); %We need to pad so that image ...
            boundary pixels are contained in a block
53
54      %Preallocate Orientations
55      Orientations = zeros(sz);
56
57      %Some parameters
58      %-Bottom of block will be the same as center before pad
59      %-Top will be bottom + block size - 1 (inclusive
60      rowHigh = row+blksz(1)-1;
61      colHigh = col+blksz(2)-1;
62      center  = padAmount+1; %Center of small block
63
64      %Now the engine, we will loop over the image, create each local block
65      %of pixels that are touching the index of interest. Do a connected
66      %components analysis, remove pixels not connected to the center pixel
67      %and then calculate orientation.
68
69      %Start the engine!
70      for ii = 1:npts
71          %Extract small block
72          block = skelPad(row(ii):rowHigh(ii),col(ii):colHigh(ii));
73
74          %Label and calculate orientation
75          Label = bwlabel(block);
76          center_label = Label==Label(center(1),center(2)); %only label ...
                of center pixel
77          rp    = regionprops(center_label,'Orientation');
78
79          %Set orientation of the center pixel equal to the calculated one
80          Orientations(row(ii),col(ii)) = rp.Orientation;
81      end
82
83  end
```

```
1   function Par = CircleFitByPratt(XY)
2
3   %--------------------------------------------------------------------------
4   %
5   %     Circle fit by Pratt
6   %      V. Pratt, "Direct least-squares fitting of algebraic surfaces",
7   %      Computer Graphics, Vol. 21, pages 145-152 (1987)
8   %
```

```
%     Input:  XY(n,2) is the array of coordinates of n points ...
    x(i)=XY(i,1), y(i)=XY(i,2)
%
%     Output: Par = [a b R] is the fitting circle:
%                           center (a,b) and radius R
%
%     Note: this fit does not use built-in matrix functions (except ...
    "mean"),
%           so it can be easily programmed in any programming language
%
%--------------------------------------------------------------------------

n = size(XY,1);      % number of data points

centroid = mean(XY);   % the centroid of the data set

%     computing moments (note: all moments will be normed, i.e. divided ...
    by n)

Mxx=0; Myy=0; Mxy=0; Mxz=0; Myz=0; Mzz=0;

for i=1:n
    Xi = XY(i,1) - centroid(1);  %  centering data
    Yi = XY(i,2) - centroid(2);  %  centering data
    Zi = Xi*Xi + Yi*Yi;
    Mxy = Mxy + Xi*Yi;
    Mxx = Mxx + Xi*Xi;
    Myy = Myy + Yi*Yi;
    Mxz = Mxz + Xi*Zi;
    Myz = Myz + Yi*Zi;
    Mzz = Mzz + Zi*Zi;
end

Mxx = Mxx/n;
Myy = Myy/n;
Mxy = Mxy/n;
Mxz = Mxz/n;
Myz = Myz/n;
Mzz = Mzz/n;

%    computing the coefficients of the characteristic polynomial

Mz = Mxx + Myy;
Cov_xy = Mxx*Myy - Mxy*Mxy;
Mxz2 = Mxz*Mxz;
Myz2 = Myz*Myz;

A2 = 4*Cov_xy - 3*Mz*Mz - Mzz;
A1 = Mzz*Mz + 4*Cov_xy*Mz - Mxz2 - Myz2 - Mz*Mz*Mz;
```

```
A0 = Mxz2*Myy + Myz2*Mxx - Mzz*Cov_xy - 2*Mxz*Myz*Mxy + Mz*Mz*Cov_xy;
A22 = A2 + A2;

epsilon=1e-12;
ynew=1e+20;
IterMax=20;
xnew = 0;

%    Newton's method starting at x=0

for iter=1:IterMax
    yold = ynew;
    ynew = A0 + xnew*(A1 + xnew*(A2 + 4.*xnew*xnew));
    if (abs(ynew)>abs(yold))
        disp('Newton-Pratt goes wrong direction: |ynew| > |yold|');
        xnew = 0;
        break;
    end
    Dy = A1 + xnew*(A22 + 16*xnew*xnew);
    xold = xnew;
    xnew = xold - ynew/Dy;
    if (abs((xnew-xold)/xnew) < epsilon), break, end
    if (iter >= IterMax)
        disp('Newton-Pratt will not converge');
        xnew = 0;
    end
    if (xnew<0.)
        fprintf(1,'Newton-Pratt negative root:  x=%f\n',xnew);
        xnew = 0;
    end
end

%    computing the circle parameters

DET = xnew*xnew - xnew*Mz + Cov_xy;
Center = [Mxz*(Myy-xnew)-Myz*Mxy , Myz*(Mxx-xnew)-Mxz*Mxy]/DET/2;

Par = [Center+centroid , sqrt(Center*Center'+Mz+2*xnew)];

end    %    CircleFitByPratt
```

```
function []=do_plot(name,k,defect,folder,corner,axi)

    clf;
    %namestr = inputname(1);
    namestr = inputname(1);
    A = cell2mat(name(k,:))';

```

```
8       if contains(folder,'config_9') == 1 || contains(folder,'config_6') ...
            == 1 || contains(folder,'config_10') == 1
9           if corner == 1
10              num = 4;
11          else
12              num=16;
13          end
14      else
15          if corner == 1
16              num = 2;
17          else
18              num=5;
19          end
20      end
21
22      %creative way to make repeated order vector
23      X = 1:1:numel(A)/num;
24      X = repmat(X,num,1);
25      X = reshape(X,[],1);
26
27      mi = min(A);
28      ma = max(A);
29
30      B = zeros(numel(A),1);
31
32      %feature scalling
33      for i=1:numel(A)
34          B(i) = (A(i)-mi)/(ma-mi);
35      end
36
37      id = find(defect == 1);
38      idx = find(defect == 0);
39
40      plot(X(idx), B(idx), 'bx')
41      hold on;
42      plot(X(id), B(id), 'ro')
43      legend('sem defeito','com defeito')
44      %text(X, B, cellstr(num2str(defect(1:numel(B)))));
45      xlabel('numero da imagem')
46      ylabel('valor da metrica')
47      hold off;
48
49      if nargin == 6
50          axis([1 numel(X) 0 axi]);
51      end
52
53      export_fig(sprintf('output_samp/%s/%s',folder,namestr),'-png','-r96');
54      fprintf('plot\n');
55  end
```

```
56
57 %y1 = 1;
58 %y2 = 19;
59 %line([0,90],[y1,y1])
60 % line([0,90],[y2,y2])
```

```
1 function [mse_out] = do_mse(distout)
2     %calculate mse based on a distances vector (distout)
3     n = numel(distout);
4     mse_out = 1/n*sum(distout.^2);
5 end
```

```
1  function [par] = pnr(metric,defect,bandwidth)
2
3      %remove nans
4      idn = find(isnan(metric));
5      metric(idn) = [];
6      defect(idn) = [];
7
8      %metric, defect given as column vectors
9      %bandwidth given as range vector [a b]
10     id = find(defect);
11     idx = find(¬defect);
12     %defect class as good
13     FP = numel(find(bandwidth(1)≤metric(id) & metric(id)≤bandwidth(2)));
14     %defect class as defect
15     TP = numel(metric(id)) - FP;
16     %good class as good
17     TN = numel(find(bandwidth(1)≤metric(idx) & metric(idx)≤bandwidth(2)));
18     %good class as good
19     FN = numel(metric(idx))-TN;
20
21     tpr = TP/(TP + FN);
22     %specificity
23     tnr = TN/(TN + FP);
24     %accuracy
25     acc = (TP + TN)/(TP + TN + FP + FN);
26     %balanced accuracy
27     bacc = (tpr + tnr)/2;
28     %precision
29     ppv = TP/(TP + FP);
30     %recall
31     rec = TP/(TP + FN);
32
33     par = [acc bacc ppv rec tnr];
34 end
```

```
1 function out = getvarname(var)
2     %this is needed to workaround flaw in matlab
3     out = inputname(1);
4 end
```

Anexo B

Resultados da peça 0 configuração 3

files	defect	line_mse	ori_mse	line_mean	line_std	line_sign	line_outliers	harris
101_exp.0.04ms.png	false	0.10298	55.737	0.23425	0.21976	0	0	1
101_exp.0.04ms.png	false	0.027344	24.772	0.027344	0.1634	0	14	5
101_exp.0.04ms.png	false	0.2681	30.965	0.43301	0.28446	0	0	0
101_exp.0.04ms.png	false	0.097534	12.386	0.27016	0.15699	0	0	0
101_exp.0.04ms.png	false	0.17121	24.676	0.17121	0.37742	0	10	1
102_0.5ms.png	false	0.031008	18.435	0.031008	0.17368	0	0	0
102_0.5ms.png	true	0.54988	18.435	0.64972	0.3581	0	16	0
102_0.5ms.png	false	0.25211	6.145	0.41877	0.27756	0	0	0
102_0.5ms.png	false	0.11709	30.725	0.28745	0.186	0	0	0
102_0.5ms.png	false	0.2607	24.676	0.2607	0.43987	0	134	0
103_1.0ms.png	false	0.011628	6.145	0.011628	0.10741	0	6	0
103_1.0ms.png	true	0.046512	24.58	0.046512	0.211	0	24	1
103_1.0ms.png	false	0.23352	12.29	0.39849	0.27389	0	10	0
103_1.0ms.png	false	0.15821	30.725	0.32579	0.22864	0	0	0
103_1.0ms.png	false	0.027237	12.338	0.027237	0.16309	0	0	0
104_1.25ms.png	false	0.0077519	6.145	0.0077519	0.087874	0	4	0
104_1.25ms.png	true	83.807	24.58	7.9001	4.6345	0	6	1
104_1.25ms.png	false	0.23429	36.87	0.39389	0.28186	0	0	0
104_1.25ms.png	false	0.092712	6.145	0.25938	0.15979	0	0	0
104_1.25ms.png	false	0.22155	55.737	0.32155	0.34441	0	18	0
105_1.5ms.png	false	0.011628	6.145	0.011628	0.10741	0	6	0
105_1.5ms.png	true	0.30274	24.58	0.46012	0.30229	0	18	0
105_1.5ms.png	false	0.09776	6.145	0.26443	0.16717	0	0	0
105_1.5ms.png	false	0.08454	6.145	0.25121	0.14669	0	0	0
105_1.5ms.png	false	0.14729	24.58	0.14729	0.35508	0	76	0
106_1.75ms.png	false	0.18992	49.16	0.18992	0.393	0	0	0
106_1.75ms.png	true	0.20021	43.015	0.35107	0.27796	0	12	0
106_1.75ms.png	false	0.13281	18.435	0.29961	0.20789	0	0	0
106_1.75ms.png	false	0.13743	24.58	0.2937	0.22664	0	0	0
106_1.75ms.png	false	0.046693	6.1689	0.046693	0.21139	0	24	0

107_2.0ms.png	false	0.10555	73.74	0.20922	0.24903	0	0	0
107_2.0ms.png	true	0.1209	30.725	0.27667	0.211	0	6	1
107_2.0ms.png	false	0.22548	12.29	0.38159	0.28317	0	0	0
107_2.0ms.png	false	0.10427	12.29	0.27108	0.17581	0	0	0
107_2.0ms.png	false	0.32606	49.544	0.4434	0.3605	0	0	0
108_3.0ms.png	false	0.023256	6.145	0.023256	0.15101	0	0	0
108_3.0ms.png	true	0.15681	30.725	0.32708	0.22365	0	0	0
108_3.0ms.png	false	0.098706	6.145	0.26536	0.16852	0	0	0
108_3.0ms.png	false	0.084206	18.435	0.25094	0.146	0	0	0
108_3.0ms.png	false	0.080439	12.338	0.24617	0.14113	0	0	0
109_4.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
109_4.0ms.png	true	NaN	NaN	NaN	NaN	NaN	NaN	NaN
109_4.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
109_4.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
109_4.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
110_5.0ms.png	false	0.067001	12.29	0.18123	0.18517	0	4	0
110_5.0ms.png	true	0.093742	30.725	0.26345	0.15631	0	0	0
110_5.0ms.png	false	0.20092	24.58	0.3574	0.27105	0	0	0
110_5.0ms.png	false	0.092712	6.145	0.25938	0.15979	0	0	0
110_5.0ms.png	false	0.16406	43.351	0.16406	0.37106	0	0	0
111_6.0ms.png	false	0.085271	30.725	0.085271	0.27983	0	0	0
111_6.0ms.png	true	0.13844	43.015	0.30911	0.20751	0	0	0
111_6.0ms.png	false	0.13954	30.725	0.30379	0.2178	0	0	0
111_6.0ms.png	false	0.11582	18.435	0.28311	0.18924	0	0	0
111_6.0ms.png	false	0.07393	18.507	0.07393	0.26217	0	2	0
112_7.0ms.png	false	0.041999	6.1689	0.1516	0.13816	0	0	0
112_7.0ms.png	true	0.083509	6.1689	0.25017	0.14493	0	0	0
112_7.0ms.png	false	0.17582	37.013	0.33782	0.24888	0	0	0
112_7.0ms.png	false	0.08796	6.1689	0.25462	0.15237	0	0	0
112_7.0ms.png	false	0.22533	18.364	0.35645	0.3141	0	18	0
113_8.0ms.png	false	0.065742	6.145	0.18763	0.17508	0	0	0
113_8.0ms.png	true	0.15279	43.015	0.31661	0.22969	0	0	0
113_8.0ms.png	false	0.10141	12.29	0.26962	0.1698	0	0	0
113_8.0ms.png	false	0.11352	18.435	0.28165	0.18529	0	0	0
113_8.0ms.png	false	0.070267	30.965	0.20689	0.16604	0	0	0
114_9.0ms.png	false	0.073643	18.435	0.073643	0.2617	0	38	0
114_9.0ms.png	true	0.16615	24.58	0.32613	0.245	0	0	0
114_9.0ms.png	false	0.11169	24.58	0.28531	0.17438	0	0	0
114_9.0ms.png	false	0.13386	6.145	0.30051	0.20909	0	0	0
114_9.0ms.png	false	0.076865	6.193	0.18548	0.20647	0	0	0
115_10.0ms.png	false	0.077821	18.507	0.077821	0.26841	0	40	0
115_10.0ms.png	true	0.16915	12.338	0.33061	0.24511	0	0	0
115_10.0ms.png	false	0.11295	24.676	0.28576	0.17723	0	0	0
115_10.0ms.png	false	0.14651	6.1689	0.31353	0.21999	0	10	0
115_10.0ms.png	false	0.065637	6.1213	0.065637	0.24813	0	0	0

Anexo C

Resultados da peça 2 configuração 3

files	defect	line_mse	ori_mse	line_mean	line_std	line_sign	line_outliers	harris
101_exp._0.04ms.png	false	0.46063	268.71	0.43701	0.52031	6	68	2
101_exp._0.04ms.png	false	0.29881	156.04	0.44627	0.3163	0	4	0
101_exp._0.04ms.png	false	0.17918	112.35	0.34295	0.24861	0	12	1
101_exp._0.04ms.png	false	0.15053	112.35	0.30865	0.23554	0	0	3
101_exp._0.04ms.png	false	0.34643	81.463	0.46483	0.36177	0	54	1
102_0.5ms.png	false	0.12933	68.39	0.25082	0.25823	0	8	0
102_0.5ms.png	false	0.20061	62.173	0.3659	0.25882	0	0	0
102_0.5ms.png	false	0.10652	43.521	0.27112	0.18206	0	0	0
102_0.5ms.png	false	0.18135	99.476	0.3304	0.26921	0	32	0
102_0.5ms.png	false	0.19566	62.664	0.36665	0.24792	0	4	0
103_1ms.png	false	0.29465	49.738	0.42135	0.34289	0	26	0
103_1ms.png	false	0.17788	62.173	0.353	0.23124	0	0	0
103_1ms.png	false	0.10858	43.521	0.27442	0.18276	0	0	0
103_1ms.png	false	0.10781	93.259	0.26961	0.18776	0	0	0
103_1ms.png	false	0.19927	56.176	0.33795	0.29222	0	0	0
104_1.25ms.png	false	0.32634	62.173	0.39553	0.413	0	58	0
104_1.25ms.png	false	0.19687	80.825	0.35748	0.26335	0	0	0
104_1.25ms.png	false	0.087127	24.869	0.25585	0.14749	0	0	0
104_1.25ms.png	false	0.11504	31.086	0.28378	0.18614	0	0	0
104_1.25ms.png	false	0.25774	37.45	0.39298	0.32205	0	10	0
105_1.5ms.png	false	0.091758	37.304	0.21532	0.21348	0	0	0
105_1.5ms.png	false	0.16732	55.955	0.32504	0.24881	0	0	0
105_1.5ms.png	false	0.095192	31.086	0.25734	0.17053	0	0	0
105_1.5ms.png	false	0.088759	18.652	0.25298	0.15765	0	0	0
105_1.5ms.png	false	0.13057	24.967	0.29251	0.21257	0	0	0
106_1.75ms.png	false	0.068201	24.869	0.18433	0.18536	0	0	0
106_1.75ms.png	false	0.14341	49.738	0.30243	0.22836	0	6	0
106_1.75ms.png	false	0.093759	24.869	0.26134	0.15987	0	0	0
106_1.75ms.png	false	0.087727	18.652	0.249	0.1607	0	0	0
106_1.75ms.png	false	0.11576	31.332	0.27391	0.20224	0	0	0

107_2ms.png	false	0.20083	24.869	0.35055	0.27973	0	12	0
107_2ms.png	false	0.11454	43.521	0.27947	0.19126	0	0	0
107_2ms.png	false	0.09612	24.869	0.25666	0.17426	0	0	0
107_2ms.png	false	0.087015	18.652	0.24935	0.15792	0	0	0
107_2ms.png	false	0.37491	18.725	0.4616	0.40308	0	50	0
108_2.25ms.png	false	0.11321	24.967	0.23654	0.23977	0	10	0
108_2.25ms.png	false	0.094696	18.725	0.25364	0.17459	0	0	0
108_2.25ms.png	false	0.10042	37.45	0.27202	0.16287	0	0	0
108_2.25ms.png	false	0.098349	18.725	0.2713	0.15763	0	0	0
108_2.25ms.png	false	0.22757	49.544	0.31719	0.35701	0	32	0
109_2.5ms.png	false	0.088435	37.304	0.22299	0.19714	0	0	0
109_2.5ms.png	false	0.10029	24.869	0.27245	0.16175	0	0	0
109_2.5ms.png	false	0.094254	24.869	0.26068	0.16249	0	0	0
109_2.5ms.png	false	0.090434	18.652	0.25166	0.16495	0	0	0
109_2.5ms.png	false	0.15611	31.332	0.29436	0.26407	0	0	0
110_3ms.png	false	0.089558	37.304	0.21141	0.21223	0	10	0
110_3ms.png	false	0.17088	24.869	0.34138	0.23356	0	0	0
110_3ms.png	false	0.089067	24.869	0.24925	0.16446	0	0	0
110_3ms.png	false	0.090273	18.652	0.25064	0.16601	0	0	0
110_3ms.png	false	0.21236	18.652	0.34808	0.30258	0	0	0
111_5ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
111_5ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
111_5ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
111_5ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
111_5ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Anexo D

Resultados da peça 4 configuração 3

files	defect	line_mse	ori_mse	line_mean	line_std	line_sign	line_outliers	harris
101_exp._1.0ms.png	false	0.22879	24.967	0.36133	0.31404	0	14	0
101_exp._1.0ms.png	false	0.098682	31.209	0.26528	0.16858	0	0	0
101_exp._1.0ms.png	false	0.0989	31.209	0.25894	0.17881	0	0	0
101_exp._1.0ms.png	false	0.11425	31.209	0.27273	0.20007	0	0	0
101_exp._1.0ms.png	false	1.3754	31.209	0.72219	0.92585	0	140	0
102_2.0ms.png	false	0.41988	31.209	0.46952	0.44746	0	44	0
102_2.0ms.png	false	0.10794	31.209	0.27456	0.1808	0	0	0
102_2.0ms.png	false	0.11205	43.692	0.27558	0.19038	0	0	0
102_2.0ms.png	false	0.086415	31.209	0.24676	0.16008	0	0	0
102_2.0ms.png	false	0.16796	37.599	0.33337	0.23886	0	0	0
103_3.0ms.png	false	0.60074	31.209	0.48233	0.60791	0	78	0
103_3.0ms.png	false	0.082315	12.483	0.24667	0.14682	0	0	0
103_3.0ms.png	false	0.093202	18.725	0.26298	0.15537	0	0	0
103_3.0ms.png	false	0.093398	21.846	0.26544	0.15175	0	0	0
103_3.0ms.png	false	0.37152	40.571	0.44638	0.41587	0	58	0
104_4.0ms.png	false	0.25408	24.967	0.37537	0.33708	0	26	0
104_4.0ms.png	false	0.096859	18.725	0.25582	0.17759	0	0	0
104_4.0ms.png	false	0.10558	31.209	0.27417	0.17472	0	0	0
104_4.0ms.png	false	0.08173	31.209	0.24238	0.1519	0	0	0
104_4.0ms.png	false	0.59573	37.599	0.52675	0.56527	0	62	0
105_5.0ms.png	false	0.27771	24.967	0.37817	0.36774	0	40	0
105_5.0ms.png	false	0.083648	15.604	0.24993	0.14583	0	0	0
105_5.0ms.png	false	0.08127	15.604	0.24587	0.14456	0	0	0
105_5.0ms.png	false	0.09817	37.45	0.26853	0.16175	0	0	0
105_5.0ms.png	false	0.10074	37.45	0.26079	0.18127	0	0	0
106_6.0ms.png	false	0.14008	12.483	0.27189	0.25772	0	0	0
106_6.0ms.png	false	0.092045	18.725	0.25986	0.15688	0	0	0
106_6.0ms.png	false	0.14465	31.209	0.30869	0.22261	0	14	0
106_6.0ms.png	false	0.0955	24.967	0.26233	0.16367	0	0	0
106_6.0ms.png	false	0.15967	37.45	0.32329	0.23532	0	0	0

107_7.0ms.png	false	0.33547	24.967	0.47697	0.32923	0	24	0
107_7.0ms.png	false	0.091972	18.725	0.24952	0.17271	0	0	0
107_7.0ms.png	false	0.11987	31.209	0.28691	0.19417	0	0	0
107_7.0ms.png	false	0.087647	18.725	0.24895	0.16054	0	0	0
107_7.0ms.png	false	0.21056	43.521	0.343	0.30542	0	42	0
108_8.0ms.png	false	0.11256	18.725	0.25067	0.22344	0	0	0
108_8.0ms.png	false	0.092099	18.725	0.25989	0.15701	0	0	0
108_8.0ms.png	false	0.080971	12.483	0.24532	0.14446	0	0	0
108_8.0ms.png	false	0.087267	24.967	0.25045	0.15696	0	0	0
108_8.0ms.png	false	0.21026	37.599	0.34199	0.30607	0	16	0
109_9.0ms.png	false	0.083621	12.483	0.24932	0.14679	0	0	0
109_9.0ms.png	false	0.091254	18.725	0.26056	0.15315	0	0	0
109_9.0ms.png	false	0.12727	43.692	0.2862	0.2134	0	10	0
109_9.0ms.png	false	0.1087	24.967	0.28218	0.17084	0	0	0
109_9.0ms.png	false	0.27651	37.45	0.40414	0.33709	0	16	0
110_10.0ms.png	false	0.19707	37.45	0.30844	0.3199	0	24	0
110_10.0ms.png	false	0.093258	18.725	0.26373	0.15426	0	0	0
110_10.0ms.png	false	0.1695	31.209	0.33107	0.24522	0	0	0
110_10.0ms.png	false	0.095297	18.725	0.26046	0.16603	0	0	0
110_10.0ms.png	false	0.21447	44.039	0.36248	0.28881	0	10	0
111_11.0ms.png	false	0.80812	24.967	0.60017	0.67059	0	118	0
111_11.0ms.png	false	0.095415	18.725	0.25708	0.17158	0	0	0
111_11.0ms.png	false	0.12584	31.209	0.29241	0.20125	0	0	0
111_11.0ms.png	false	0.087988	18.725	0.25074	0.1588	0	0	0
111_11.0ms.png	false	0.12865	43.692	0.2924	0.20814	0	0	0
112_12.0ms.png	false	0.27137	24.967	0.39756	0.3373	0	16	0
112_12.0ms.png	false	0.1013	18.725	0.2612	0.18222	0	0	0
112_12.0ms.png	false	0.11897	31.209	0.28535	0.19416	0	0	0
112_12.0ms.png	false	0.082539	18.725	0.24337	0.15298	0	0	0
112_12.0ms.png	false	0.27368	43.865	0.39345	0.34547	0	34	0
201_1.0ms.png	false	0.12553	31.456	0.23451	0.2661	0	32	0
201_1.0ms.png	false	0.32935	6.2913	0.49602	0.28923	0	0	0
201_1.0ms.png	true	1.3945	101	0.91161	0.75215	2	16	1
201_1.0ms.png	false	0.31007	9.4369	0.47674	0.28829	0	0	0
201_1.0ms.png	true	0.2464	34.33	0.30704	0.3908	0	14	0
202_2.0ms.png	false	0.10184	18.799	0.2593	0.18639	0	0	0
202_2.0ms.png	false	1.3294e-24	0	9.9332e-13	5.8658e-13	0	0	0
202_2.0ms.png	true	3.0352	53.606	1.5013	0.88563	2	16	1
202_2.0ms.png	false	1.3699e-24	0	9.1298e-13	7.3382e-13	0	0	0
202_2.0ms.png	true	0.26216	44.039	0.37589	0.34835	0	14	0
203_3.0ms.png	false	0.16601	12.533	0.16601	0.37283	0	84	0
203_3.0ms.png	false	0.25392	6.2664	0.42058	0.27809	0	0	0
203_3.0ms.png	true	5.0258	83.885	1.956	1.0975	2	56	2
203_3.0ms.png	false	0.96923	62.664	0.82282	0.54163	0	0	0
203_3.0ms.png	true	0.67063	39.492	0.40873	0.71104	0	114	0

204_4.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
204_4.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
204_4.0ms.png	true	NaN	NaN	NaN	NaN	NaN	NaN	NaN
204_4.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
204_4.0ms.png	true	NaN	NaN	NaN	NaN	NaN	NaN	NaN
205_5.0ms.png	false	0.083004	6.2664	0.083004	0.27644	0	0	0
205_5.0ms.png	false	6.3585e-24	0	2.1838e-12	1.2632e-12	0	0	0
205_5.0ms.png	true	0.098814	33.07	0.051383	0.31073	2	16	1
205_5.0ms.png	false	6.124e-24	0	2.1291e-12	1.2639e-12	0	0	0
205_5.0ms.png	true	0.22004	33.201	0.24751	0.39927	0	14	0
206_6.0ms.png	false	0.18583	43.865	0.32	0.28941	0	18	0
206_6.0ms.png	false	0.3205	12.533	0.47545	0.30793	0	0	0
206_6.0ms.png	true	0.25345	66.139	0.40614	0.29808	2	10	1
206_6.0ms.png	false	6.4614e-24	0	2.1656e-12	1.3337e-12	0	0	0
206_6.0ms.png	true	0.27063	47.185	0.40958	0.32138	0	30	1
207_7.0ms.png	false	0.19368	12.533	0.16206	0.40997	0	8	0
207_7.0ms.png	false	5.7183e-24	0	2.0542e-12	1.2266e-12	0	0	0
207_7.0ms.png	true	2.7913	26.803	1.4423	0.84489	0	12	0
207_7.0ms.png	false	6.4614e-24	0	2.1656e-12	1.3337e-12	0	0	0
207_7.0ms.png	true	0.19654	47.185	0.32443	0.30275	0	6	1
208_8.0ms.png	false	0.068095	6.2664	0.2171	0.14507	0	0	0
208_8.0ms.png	false	5.7183e-24	0	2.0542e-12	1.2266e-12	0	0	0
208_8.0ms.png	true	3.7077	18.799	1.6714	0.95806	0	14	1
208_8.0ms.png	false	6.4614e-24	0	2.1656e-12	1.3337e-12	0	0	0
208_8.0ms.png	true	0.33411	34.602	0.33245	0.47379	0	16	0
209_9.0ms.png	false	0.059289	6.2664	0.059289	0.23663	0	30	0
209_9.0ms.png	false	5.7183e-24	0	2.0542e-12	1.2266e-12	0	0	0
209_9.0ms.png	true	2.6476	12.533	1.3847	0.85623	0	0	2
209_9.0ms.png	false	42.013	43.865	5.6145	3.2454	0	0	0
209_9.0ms.png	true	0.16276	41.056	0.26388	0.30578	0	6	1
210_10.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
210_10.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
210_10.0ms.png	true	NaN	NaN	NaN	NaN	NaN	NaN	NaN
210_10.0ms.png	false	NaN	NaN	NaN	NaN	NaN	NaN	NaN
210_10.0ms.png	true	NaN	NaN	NaN	NaN	NaN	NaN	NaN
211_11.0ms.png	false	0.13613	34.465	0.274	0.24759	0	0	0
211_11.0ms.png	false	0.63622	40.732	0.67435	0.42684	0	8	0
211_11.0ms.png	true	0.2207	81.463	0.3602	0.30219	0	10	0
211_11.0ms.png	false	1.3969e-24	0	1.0152e-12	6.0634e-13	0	0	0
211_11.0ms.png	true	1.1204	53.265	0.63582	0.84794	0	104	1
212_12.0ms.png	false	0.1502	12.533	0.14229	0.3612	0	70	0
212_12.0ms.png	false	1.6499e-24	0	1.1065e-12	6.5363e-13	0	0	0
212_12.0ms.png	true	3.9762	25.066	1.7278	0.9974	0	14	0
212_12.0ms.png	false	1.3969e-24	0	1.0152e-12	6.0634e-13	0	0	0
212_12.0ms.png	true	0.18403	40.893	0.27918	0.32636	0	10	1