CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

COMPARAÇÃO DE TÉCNICAS DE CORRESPONDÊNCIA DE PONTOS CHAVE PARA ALGORITMO ORB

BRUNO FELIPE DE NADAI DA SILVA

ORIENTADOR: PROF. ME. RODOLFO BARROS CHIARAMONTE

Marília - SP
Dezembro/2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

COMPARAÇÃO DE TÉCNICAS DE CORRESPONDÊNCIA DE PONTOS CHAVE PARA ALGORITMO ORB

BRUNO FELIPE DE NADAI DA SILVA

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.
Orientador: Prof. Me. Rodolfo Barros Chiaramonte

Marília - SP
Dezembro/2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Bruno Felipe de Nadai da Silva

COMPARAÇÃO DE TÉCNICAS DE CORRESPONDÊNCIA DE PONTOS CHAVE PARA ALGORITMO ORB

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9,5 (nove e meio)

Orientador: Rodolfo Barros Chiaramonte

1º.Examinador: Mauricio Duarte

2º.Examinador: Luan Cardoso dos Santos

Marília, 28 de novembro de 2017.

Dedico este trabalho a meu pai Marcos Rogério e à minha avó Maria de Lourdes por serem a minha base e porto seguro. À minha mãe Raquel Borges que apesar das desventuras passadas, faz o seu máximo para estar presente. Às minhas irmãs Eloísa e Lara por me fornecerem momentos de sorrisos e ternura que somente a família pode oferecer.

A todos os meus colegas de sala, em especial ao Fernando Franco, por ter sempre chamado minha atenção em sala de aula nos meus momentos de distração e procrastinação.

Agradeço a todos os professores que lecionaram no curso de Ciência da Computação durante a minha graduação, em especial, ao meu orientador Prof. Me. Rodolfo Barros Chiaramonte, pelo auxílio e orientação durante a produção deste trabalho.

A persistência é o caminho do êxito
Charles Chaplin

RESUMO

Na visão computacional, a técnica de correspondência de pontos chave (*Feature Matching*) é uma das bases da resolução de problemas como reconhecimento e rastreamento de objetos, fotografias panorâmicas, entre outros. Neste processo, é necessário antes realizar a detecção e a descrição dos pontos chave, assim, vários estudos foram conduzidos sobre detecção e descrição de pontos chave e dessa maneira, vários algoritmos foram desenvolvidos, como SIFT, SURF, ORB, entre outros. Em aplicações que necessitam de execução em tempo real, é necessário selecionar algoritmos de alto desempenho. O algoritmo ORB tem se destacado em algoritmos de tempo real e portanto, foi selecionado como detector e descritor base deste trabalho. Múltiplos algoritmos foram desenvolvidos para realizar a correspondência de pontos chave utilizando força bruta, raio de distância e árvores k-d. A fim de embasar a decisão sobre qual método de correspondência utilizar em aplicações de tempo real, este trabalho implementa e analisa o desempenho de três dos algoritmos mais populares, *Brute Force Matching*, *K Nearest Neighbors* e a biblioteca *Fast Library for Approximate Nearest Neighbors*.

Palavras-chave: Feature Matching, Bechmarking, SIFT, SURF, ORB, BFM, KNN, FLANN

ABSTRACT

In computer vision, the feature matching technique is one of the bases for solving problems like object detection and tracking, panoramic photographies and others. For this process, must first detect and describe the features, to do the feature matching. Many researchs has been made on feature detection and feature description (bases for feature matching), causing many algorithms to be developed, like SIFT, SURF and ORB. On real time applications, an an optimum algorithm with high performance must be choosen. The ORB algorithm has been increasing its popularity among detection and description algorithms for its performance, beign the chosen algorithm for this paper. Many feature matching algorithms were developed using techniques like brute force, distance ratio and k-d trees. In order to base the choice of which matching method should be used on real time applications, this paper implements and analyzes the performance of three of the most popular algorithms, the Brute Force Matching, the K Nearest Neighbors and the Fast Library for Approximate Nearest Neighbors.

Palavras-chave: Feature Matching, Bechmarking, SIFT, SURF, ORB, BFM, KNN, FLANN

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

VANT	- *Veículo Aéreo Não Tripulado*
ORB	- *Oriented Fast and Rotated Brief*
BRIEF	- *Binary Robust Independent Elementary Features*
SIFT	- *Scale Invariant Feature Transform*
SURF	- *Speed Up Robust Features*
BFM	- *Brute Force Matcher*
NN	- *Nearest Neighbors*
KNN	- *K Nearest Neighbors*
SQT	- Soma dos Quadrados Totais
SQD	- Soma dos Quadrados de Dentro
SQE	- Soma dos Quadrados Entre

SUMÁRIO

Capítulo 1
INTRODUÇÃO

Na visão computacional, um dos problemas emergentes (RUBLEE et al., 2011) é o desempenho dos algoritmos selecionados para realização de funções dependentes de visão, principalmente em sistemas *real time* (em tempo real), uma vez que algoritmos de processamento custoso influenciam diretamente no resultado dos mesmos. Um exemplo de sistema *real time* é o uso da visão computacional para localização e reconhecimento utilizando VANT's (Veículos Aéreos Não Tripulados — normalmente drones), cuja resposta dos algoritmos precisa ser a mais rápida possível, uma vez que atrasos no processamento podem causar problemas como colisão e, consequentemente, queda.

Uma dessas funções dependentes de visão é a técnica de correspondência de pontos chave (*Feature Matching*) entre imagens, sendo uma das bases para resolver vários problemas como reconhecimento de objetos (*Object Recognition*), aquisição de estrutura tridimensional a partir de movimento (*Structure From Motion*), odometria digital (*Digital Odometry*), rastreamento de objetos (*Object Tracking*), entre outros.

A correspondência de pontos chave é executada sobre os dados obtidos de dois processos chamados **detecção dos pontos chave** (*Feature Detection*) e **descrição dos pontos chave** (*Feature Description*), assim, para encontrar pontos chave correspondentes entre duas imagens, é necessário detectá-los e descrever suas características.

Rublee et al. (2011) propôs um método chamado ORB (*Oriented fast and Rotated Brief*), que realiza a detecção e descrição de pontos chave em uma imagem de maneira ágil. O algoritmo ORB ganhou notoriedade por sua capacidade de encontrar bons pontos chave e descrevê-los de maneira ágil, sendo muito útil em sistemas *real time*.

Entretanto, em sistemas *real time*, os algoritmos que realizam correspondência destes pontos ainda são, em sua grande maioria, selecionados de maneira empírica,

sem embasamento teórico.

Este trabalho visa comparar o desempenho entre técnicas de correspondência de pontos chave quando aplicadas à pontos chave descritos pelo algoritmo ORB, de maneira a oferecer embasamento teórico para a seleção de um algoritmo de correspondência.

1.1 Contexto

Rublee et al. (2011) propôs um método de detecção e descrição de pontos chave, construído utilizando como base o algoritmo de detecção FAST (ROSTEN; DRUMMOND, 2006) e o algoritmo de descrição BRIEF (CALONDER et al., 2010). Seu trabalho adicionou técnicas de orientação para o detector FAST, a computação de pontos chave BRIEF orientadas, com análise de variação e correlação, além de um modelo de aprendizado de des-correlação de pontos chave BRIEF com invariância rotacional que permite melhor performance em detecções do tipo vizinho mais próximo (*nearest neighbor*), dando assim o nome de ORB (*Oriented-FAST and Rotated BRIEF*) para o algoritmo.

O método ORB se comporta tão bem quanto o algoritmo SIFT (LOWE, 2004), mas executa até duas vezes mais rápido em muitas situações, sendo desta maneira, um algoritmo detector e descritor de pontos chave com um ótimo desempenho para em aplicações em tempo real (RUBLEE et al., 2011) .

1.2 Motivação e Objetivos

Ao utilizar o algoritmo ORB para detecção e descrição dos pontos chave em aplicações, ainda faz-se necessário escolher um algoritmo para realizar a correspondência dos pontos fornecidos pelo ORB.

Esta pesquisa tem como objetivo avaliar e comparar o desempenho entre as técnicas de correspondência de pontos chave *Brute Force Matcher* (Correspondência por Força Bruta), *K Nearest Neighbor* (N Vizinhos Próximos) e *Fast Library for Approximate Nearest Neighbors* (Biblioteca Ágil para Vizinhos Aproximados), utilizando o algoritmo ORB como detector e descritor base. Desta maneira, este trabalho auxiliará na eleição de uma algoritmo de correspondência para o algoritmo ORB, sendo esta necessária em muitos projetos como reconhecimento facial, fotografias panorâmicas, acompanhamento de objetos, entre tantas outras.

Assim, tem como objetivo geral avaliar e analisar o desempenho dos algoritmos BFM, KNN e FLANN de correspondência de pontos chave utilizando o ORB como detector e descritor.

1.3 Materiais e Métodos

A lista abaixo descreve os passos necessários para realização desta pesquisa:

- Revisão bibliográfica sobre os conceitos e técnicas de detecção, descrição e correspondência de pontos chave;
- Implementação do algoritmo ORB utilizando a biblioteca OpenCV 3.0.0 para Python 2.7, e aplicar em imagens e vídeos;
- Implementação dos algoritmos BFM, KNN e FLANN para realizar correspondência de pontos chaves encontrados pelo ORB;
- Medir o desempenho médio e o desvio padrão para cada algoritmo de correspondência para cada quadro do vídeo;
- Realizar análise comparativa de desempenho para cada algoritmo de correspondência;

1.4 Organização do Trabalho

O Capítulo 2 é referente à técnicas de detecção e descrição de pontos chave, assim como algoritmos que realizam estas tarefas, quais problemas resolvem e quais suas aplicações. O Capítulo 3.2 aborda as técnicas de correspondência de pontos chave, como funcionam, quais as técnicas selecionadas utilizadas para esta pesquisa e o motivo das mesmas terem sido selecionadas.

No Capítulo 3 é descrito como o algoritmo ORB foi implementado em conjunto com os algoritmos de correspondência para então, descrever no Capítulo 4 a análise dos dados: medições de desempenho, medição de desvio padrão e comparativos entre algoritmos.

O Capítulo 5 contém a conclusão sobre a pesquisa assim como suas considerações finais.

Capítulo 2

DETECTORES E DESCRITORES DE PONTOS CHAVE E ALGORITMOS DE CORRESPONDÊNCIA

Na visão humana, a detecção e a descrição de pontos chave são processos importantes, podendo ser descritos como uma forma de se perceber informações geométricas e limites do ambiente e dos objetos nele contido.

Na visão computacional, pontos chave (*features*) são definidas como áreas ou pontos de interesse em uma imagem, sendo definido de acordo com o problema a ser resolvido. Bons exemplos de características são bordas, texturas e blobs (CANNY, 1986).

2.1 Detecção de Pontos Chave

Na visão computacional, a detecção de pontos chave é uma operação de baixo nível, sendo normalmente a primeira operação a ser executada, examinando cada pixel de uma imagem, verificando se o pixel atende os critérios desejados para ser elegido como um ponto chave (SMITH; BRADY, 1995).

A detecção de pontos chave é uma área de pesquisa específica na visão computacional, assim, muitos algoritmos de detecção de pontos chave foram desenvolvidos na última década. Algoritmos de detecção são comumente utilizados como parte de algoritmos maiores, permitindo que algoritmos que tem como entrada a saída dos algoritmos de detecção tenham ganho de desempenho, processando apenas em pontos e/ou áreas de interesse, definidas por estes pontos chave. A precisão desta etapa tem grande influência no sucesso (ou fracasso) dos processos de análise computadorizada (CANNY, 1986). Estes algoritmos variam muito de acordo com o tipo de pontos

chave detectados, assim como sua complexidade computacional.

A seguir, serão apresentadas algumas técnicas de detecção de pontos chave desenvolvidas ao longo do tempo.

2.1.1 Detecção de Bordas - *Edge Detector*

Uma borda (também chamada de contorno) pode ser descrita como o limite entre duas regiões com certa diferença de tons de cinza (NOVAIS, 2016). As bordas podem ser classificadas como degrau, rampa ou telhado, conforme ilustra a figura 2.1.

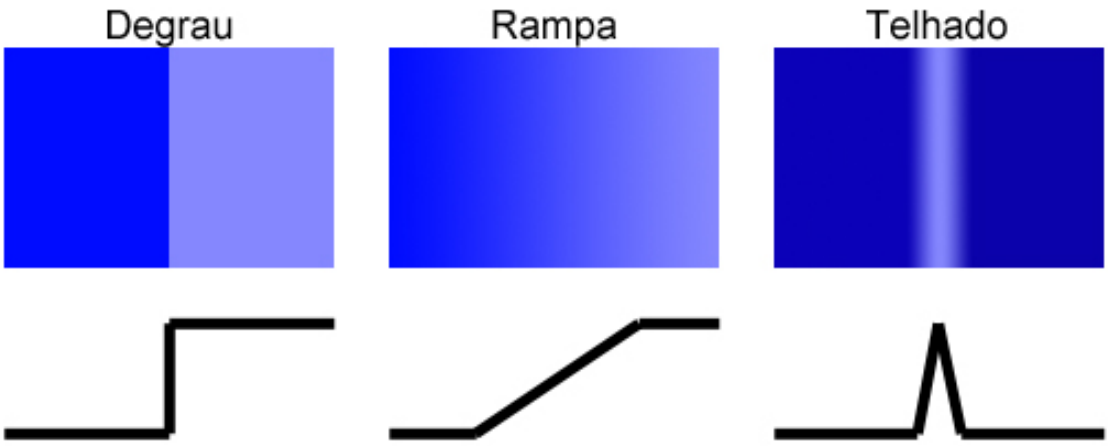

Figura 2.1 – Tipos de Bordas. Adaptado de: (GONZALES; WOODS, 2011).

Para realizar a detecção de bordas, aplica-se uma máscara, em todos os pixeis da imagem. A máscara mais comum é a máscara 3x3, representada pela figura 2.2, onde é calculado o somatório dos produtos resultantes da multiplicação de cada coeficiente da máscara pelo nível de cinza do respectivo pixel correspondente na imagem. Este procedimento (GONZALES; WOODS, 2011) também pode ser representado pela equação

$$R = \sum_{i=1}^{9} w_i z_i$$

Onde z_i é o nível de cinza associado ao coeficiente w_i da máscara, e R é o resultado obtido através da máscara utilizada.

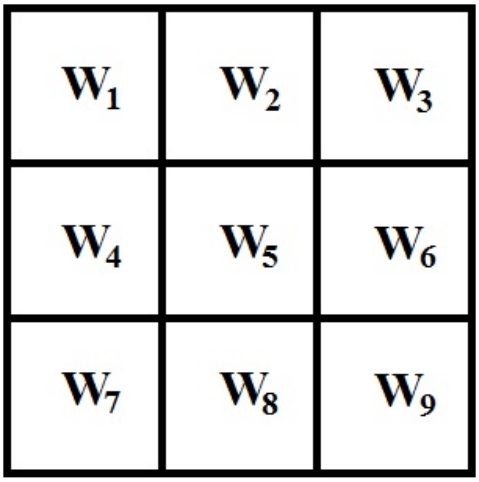

Figura 2.2 – Máscara 3x3.

Para realizar a detecção de bordas (GONZALES; WOODS, 2011), o operador de derivação local $G(x,y)$ é geralmente utilizado, sendo denotado por ∇f e dado por

$$\nabla f = \left[G_x G_y\right] = \left[\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\right]$$

De acordo com Novais (2016), uma das formas mais simples de se computar o gradiente é realizar uma aproximação digital das derivadas de primeira ordem, utilizando máscaras de tamanho 3x3, dadas pelas equações:

$$G_x = \frac{\partial f}{\partial x} = (w_7 + w_8 + w_9) - (w_1 + w_2 + w_3)$$

$$G_y = \frac{\partial f}{\partial y} = (w_3 + w_6 + w_9) - (w_1 + w_4 + w_7)$$

Onde w_i é um coeficiente de uma máscara geral 3x3, ilustrada na figura 2.2.

Estas equações são aplicadas ao longo de toda a imagem, utilizando os operadores de Prewitt, máscaras representadas pela figura 2.3.

Borda Horizontal

-1	-1	-1
0	0	0
1	1	1

Borda Vertical

-1	0	1
-1	0	1
-1	0	1

Borda em −45°

0	1	1
-1	0	1
-1	-1	0

Borda em 45°

-1	-1	0
-1	0	1
0	1	1

Figura 2.3 – Operadores de Prewitt. Adaptado de: (NOVAIS, 2016)

Uma variação permite utilizar peso 2 no coeficiente central, produzindo suavização da imagem fazendo com que haja, assim, supressão de ruído. A utilização do

peso 2 no coeficiente central gera uma pequena variação nas equações (GONZALES; WOODS, 2011), sendo:

$$G_x = \frac{\partial f}{\partial x} = (w_7 + 2w_8 + w_9) - (w_1 + 2w_2 + w_3)$$

$$G_y = \frac{\partial f}{\partial y} = (w_3 + 2w_6 + w_9) - (w_1 + 2w_4 + w_7)$$

Para implementação destas equações, utiliza-se uma variação das máscaras de Prewitt, operadores chamados de operadores de Sobel (NOVAIS, 2016), ilustrado na figura 2.4.

Borda Horizontal

-1	-2	-1
0	0	0
1	2	1

Borda Vertical

-1	0	1
-2	0	2
-1	0	1

Borda em −45°

0	1	2
-1	0	1
-2	-1	0

Borda em 45°

-2	-1	0
-1	0	1
0	1	2

Figura 2.4 – Operadores de Sobel. Adaptado de: (NOVAIS, 2016)

Um dos algoritmos mais populares para detecção de bordas é o algoritmo de Canny, que baseia-se em sólida fundamentação teórica, cuja complexidade está muito longe da trivialidade dos operadores convencionais de borda (VALE; POZ, 2004). Canny (1986) observou que muitos algoritmos existentes de detecção de bordas possuíam requisitos similares, sendo eles:

- Detecção do maior número possível de bordas na imagem;
- As bordas encontradas devem estar o mais próximo possível da borda real;
- Uma borda na imagem deve ser detectada uma única vez, e ruídos não devem gerar falso-positivos.

Dessa maneira, o desenvolvimento de um algoritmo capaz de atender estes requisitos em comum poderia ser utilizado em uma gama maior de soluções.

Canny (1986) desenvolveu então um algoritmo que, dentre os algoritmos de detecção de bordas desenvolvidos é um dos algoritmos mais rigorosamente definidos, fornecendo uma detecção com boa precisão e confiabilidade (SWETHA; VEENA; ATHAVALE, 2016).

O algoritmo desenvolvido por Canny para detecção de bordas pode ser descrito através de cinco passos, sendo estes:

(a) Uma vez que os resultados de detecção de bordas são facilmente afetados por ruídos na imagem, é de suma importância filtrar o ruído para evitar falso-positivos. Para filtrar o ruído, a imagem é ligeiramente suavizada através da aplicação de um filtro Gaussiano, reduzindo os efeitos de ruídos óbvios na imagem. O resultado deste passo é uma versão ligeiramente desfocada da imagem original.

(b) Uma borda em uma imagem pode apontar para qualquer direção, assim, o algoritmo de Canny utiliza quatro operadores (a implementação mais comum é a de Sobel) para determinar a direção do gradiente, arredondando a direção para um dos quatro ângulos, vertical, horizontal e diagonais (0^o, 90^o, 45^o e 135^o, respectivamente).

(c) Devido à suavização, as bordas extraídas dos gradientes se encontrarão desfocadas. Assim, é aplicada uma técnica de "desbaste" da borda chamada *non-maximum suppresion*, para suprimir todos os valores do gradiente para 0, exceto no local de maior valor, que indica o local com maior mudança de intensidade. Essa supressão é realizada pixel a pixel, comparando a magnitude do pixel à magnitude do gradiente nos polos positivos e negativos do operador. Caso a magnitude do pixel seja maior que a magnitude dos outros pixeis no operador de mesma direção, o valor do pixel é preservado, caso contrário, o mesmo é suprimido (seu valor é definido como 0).

(d) Após a aplicação do desbaste da borda, os pixeis remanescentes provêm uma melhor representação das bordas reais da imagem. Entretanto, algumas bordas podem ter sido causadas por ruídos e variações de cor. Para filtrar as bordas consideradas ruins, Canny aplica um limiar mínimo e um limiar máximo (comumente passado como parâmetro pela aplicação) em todos os pixeis de borda. Caso o valor de gradiente do pixel esteja acima do limiar máximo, o pixel é considerado como um pixel "forte". Caso o valor de gradiente do pixel esteja abaixo do limiar máximo, mas acima do limiar mínimo, o pixel é considerado como um pixel "fraco". Caso o valor de gradiente do pixel esteja abaixo do limiar mínimo, o pixel é suprimido.

(e) Neste momento, os pixeis marcados como "fortes" já são considerados bordas e estão incluídos no resultado. Os pixeis marcados como pixeis "fracos" podem ser provindos de bordas reais, ou ainda podem ser provindos de ruídos e alterações Para detectar quais pixeis "fracos" são bordas boas, Canny verifica se a borda fraca está conectada a uma borda forte, considerando a mesma como uma borda válida caso esteja, descartando-a caso contrário.

A figura 2.5 demonstra o resultado final da aplicação do algoritmo de Canny em uma imagem, onde é possivel verificar a eficiência do algoritmo em realizar a detecção de bordas, mesmo onde a diferença de cor é pequena, através da técnica de dupla limiariazação.

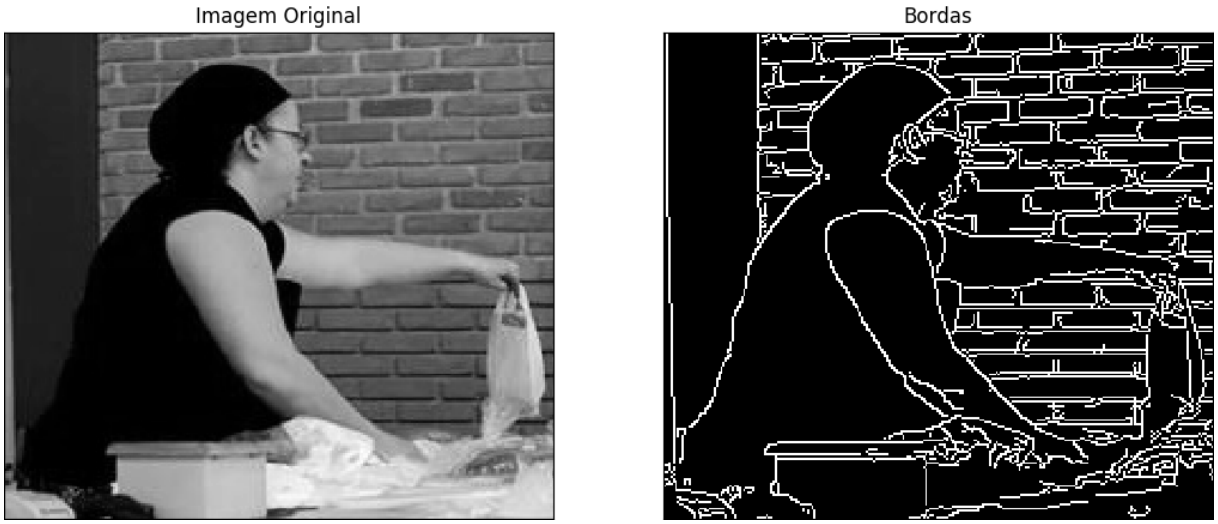

Figura 2.5 – Exemplo de detecção de Bordas utilizando o algoritmo de Canny (1986).

2.1.2 Detecção de Cantos - *Corner Detector*

Um canto pode ser definido como uma intersecção de duas bordas, ou também, como um ponto onde exista duas direções dominantes e distintas para o gradiente na sua vizinhança (ROSTEN; DRUMMOND, 2006).

Um ponto de interesse em uma imagem é um ponto que possui uma posição bem definida e pode ser detectado robustamente. Isso significa que um ponto de interesse pode tanto ser um canto quanto ser apenas um ponto isolado com intensidade local extrema, final de linhas ou pontos mal localizados na curva.

Na prática, (WILLIS; SUI, 2009) muitos dos métodos denominados detectores de cantos detectam pontos gerais de interesse, de maneira que o termo “canto” e “ponto de interesse” são usados como similares nas literaturas. Como consequência, quando apenas cantos precisam ser detectados, torna-se necessário realizar uma análise local nos pontos chave detectados para determinar quais pontos são cantos e quais devem ser descartados. Assim, detectores do tipo borda podem ser utilizados com pós-processamento para detectar apenas os cantos com um algoritmo secundário, como por exemplo, o operador de Kirsch (1971) (SHAPIRO; STOCKMAN, 2001).

Rosten e Drummond (2006) desenvolveram o FAST, um algoritmo de detecção de cantos cujo objetivo era identificar pontos chave em imagens com um melhor desempenho quando comparado ao algoritmo SIFT (LOWE, 2004). O algoritmo FAST

entretanto é altamente sensível a ruídos e não possui informação sobre a orientação dos descritores, tendo proposto como solução a utilização de aprendizado de máquina e criação de uma árvore de decisão, arvore esta que é transformada em código na linguagem C e compilado.

O algoritmo FAST executa como primeiro passo a seleção de um pixel p na imagem para ser analisado, onde define-se I_p como a intensidade do mesmo. Em seguida, é selecionado e definido (através de parâmetro) um valor apropriado para o limiar t.

Conforme ilustra a figura 2.6, cria-se um círculo de 16 pixeis ao redor do pixel p, estes pixeis serão analisados para definir se o pixel p será considerado um ponto, ou descartado.

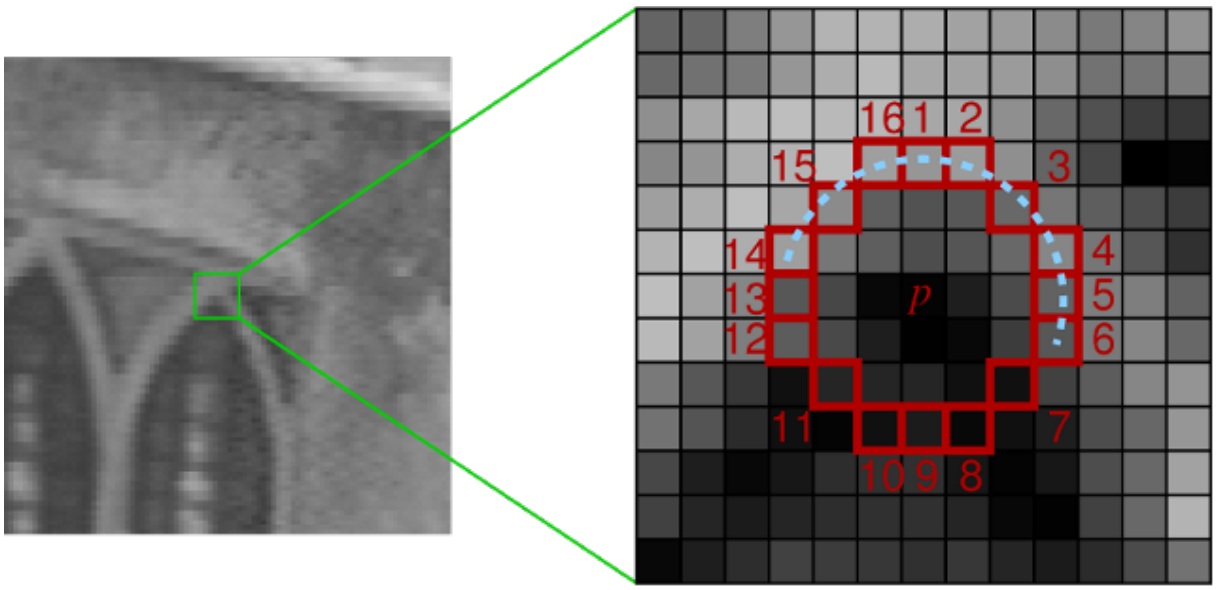

Figura 2.6 – Vizinhança circular do pixel p. Fonte: (ROSTEN; DRUMMOND, 2006)

O pixel p será considerado um canto se existir um conjunto de n (normalmente, considera-se $n >= 12$) pixeis adjacentes no círculo que sejam todos mais claros que $I_p + t$ ou todos mais escuros que $I_p - t$ (conforme ilustra a linha tracejada em azul na figura 2.6).

Uma análise de alta performance (ROSTEN; DRUMMOND, 2006) foi proposta para descartar grande parte dos pontos que não são cantos. Nesta análise, examina-se apenas os pixeis vizinhos que estão no círculo nas posições 1, 5, 9 e 13, testando primeiramente os pixeis 1 e 9 e se necessário, os pixeis 5 e 13, verificando se os mesmos são mais claros ou mais escuros que p. Para p ser considerado um canto, ao menos três destes quatro pixeis devem deve ser mais claros que $I_p + t$ ou mais escuros que $I_p - t$. Caso o critério não seja atendido, então p não é considerado

como um canto.Essa análise, apesar de boa performance, apresenta pontos negativos (ROSTEN; DRUMMOND, 2006):

- Não é muito eficaz quando o limiar $n < 12$;
- A escolha do pixel não é considerada ótima, uma vez que sua eficiência depende da distribuição e aparência dos cantos;
- Os resultados da análise são descartados, sendo acessíveis somente no momento da análise;
- Múltiplos pontos chave são detectados adjacentes.

A figura 2.7 demonstra o resultado final da aplicação do algoritmo FAST em uma imagem, onde cada ponto p detectado como canto foi demarcado por um círculo verde.

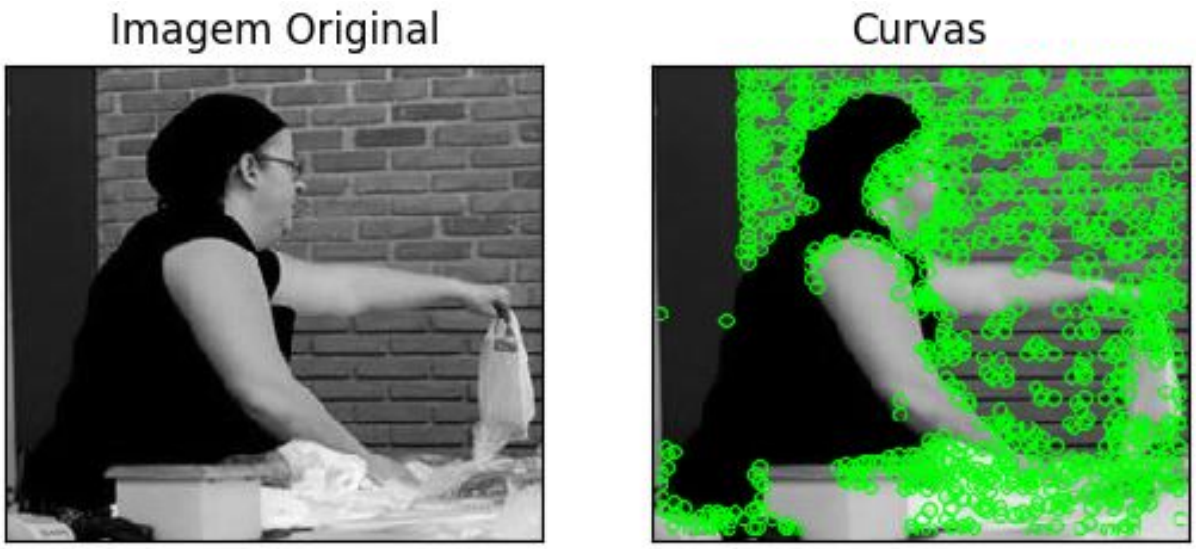

Figura 2.7 – Exemplo de detecção de cantos utilizando o algoritmo FAST de Rosten e Drummond (2006). Cada ponto indica a localização de um ponto p considerado um canto.

2.1.3 Detecção de Blob - *Blob Detector*

Na visão computacional, denomina-se blob uma região de uma imagem onde as propriedades dos pixeis são (aproximadamente) constate, ou seja, todos os pontos dentro de um blob podem ser de alguma maneira interpretados como similares (LINDEBERG, 1998). O método mais comum para detecção de blobs é através da convolução.

Considerando uma propriedade de interesse (expressa em forma de função de posição na imagem), existem dois principais tipos de detectores de blob:

1. Métodos diferenciais, baseados em derivadas da função em relação à posição;
2. Métodos baseados em extremos locais, baseados em buscar o extremo máximo e mínimo da função.

Nas terminologias mais recentes, (LINDEBERG, 2008) estes detectores podem ser referenciados também como **operadores de ponto de interesse** (*Interest Point Operators*) ou **operadores de região de interesse** (*Interest Region Operators*).

Uma das principais motivações para o estudo e desenvolvimento dos detectores do tipo *blob* é prover informações complementares sobre as regiões, não obtidas através de técnicas de detecção do tipo *Edge* ou *Corner* (LINDEBERG, 2008). Nos primórdios, a detecção de blobs era realizada como pré processamento, utilizada para obter regiões de interesse a serem processadas, uma vez que essas regiões podiam sinalizar a presença de objetos (ou parte de objetos) na imagem, sendo muito útil para reconhecimento e/ou rastreamento de objetos (LINDEBERG, 2008).

Em trabalhos mais recentes (LINDEBERG, 2013), descritores de *blob* se tornaram amplamente populares como pontos de interesse base para correspondência em visão estereoscópica, assim como para sinalizar a presença de recursos de imagem informativos para reconhecimento de objeto baseado em aparência com base em estatísticas de imagem locais.

Um dos métodos mais conhecidos para detecção de *blobs* é a extração da diferença de Gaussianos (referido como *Difference of Gaussians*). O método de diferença de Gaussianos é um método diferencial que aplica um filtro Gaussiano em uma imagem e, em seguida, subtrai essa imagem da imagem original, resultando assim em uma terceira imagem contendo apenas as diferenças, conforme pode ser visto na figura 2.8.

Assim, dada uma imagem $I : \{\mathbb{N} \subseteq \mathbb{R}^n\} \rightarrow \{\mathbb{Y} \subseteq \mathbb{R}^m\}$, a diferença de Gaussianos para a imagem I será a função $F_{\sigma 1,\sigma 2} : \{\mathbb{X} \subseteq \mathbb{R}^n\} \rightarrow \{\mathbb{Z} \subseteq \mathbb{R}\}$, obtida ao subtrair a imagem I suavizada com um filtro Gaussiano σ_2^2 da imagem I suavizada com um filtro Gaussiano σ_1^2 de menor intensidade, com $\sigma_2 > \sigma_1$, de maneira que

$$F_{\sigma 1,\sigma 2}(x) = I * \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_1^2}} - I * \frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_2^2}} \tag{2.1}$$

formada pelas equações

$$\frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_1^2}} \tag{2.2}$$

$$\frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_2^2}} \tag{2.3}$$

onde I é a imagem original, a equação 2.2 é o filtro Gaussiano σ_2^2 e a equação 2.3 é o filtro Gaussiano σ_2^1 de menor intensidade.

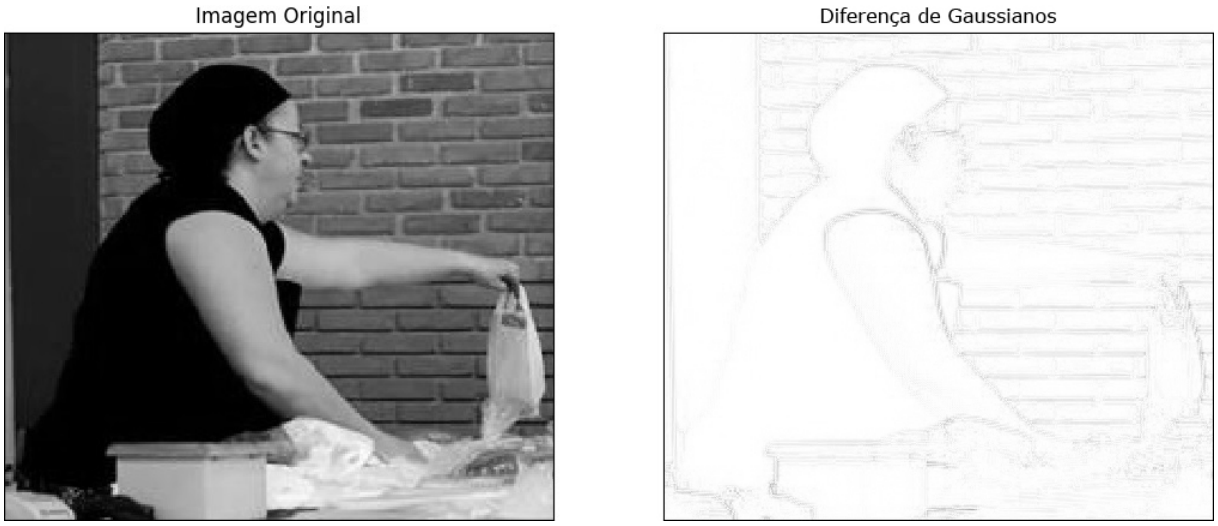

Figura 2.8 – Exemplo da aplicação da detecção de *blobs* através da técnica de diferença de Gaussianos.

2.2 Detecção e Descrição de Pontos Chave

Pontos chave detectados em imagens possuem localização bem definida e são comumente associados à significativas variações de propriedades como intensidade, cor e textura. A descrição de pontos chave é técnica que define características a estes pontos, permitindo que, além da localização de um ponto chave, também haja informações sobre o ponto. A descrição de pontos chave é utilizada como base de muitos algoritmos como reconhecimento de objetos em imagens, calibração de câmeras, descrição de características de imagens, entre outras aplicações (NOVAIS, 2016).

Para realizar a descrição dos pontos chave, é realizada a extração das características de cada ponto. Cada algoritmo existente realiza a descrição das características dos pontos chave de uma forma distinta. A seguir, serão abordados os três principais algoritmos que realizam descrição de pontos chave e que mais se destacaram ao longo do tempo.

2.2.1 SURF

O algoritmo SURF (Speeded-Up Robust Featuers), proposto por Bay, Tuytelaars e Gool (2006) é capaz de detectar, descrever e reconhecer pontos chave em uma imagem, atribuindo à cada ponto um vetor descritor. O algoritmo SURF processa a imagem em nível de cinza, não utilizando nenhum informação relacionada as cores da imagem.

As etapas do algoritmo SURF serão abordadas a seguir.

2.2.1.1 Detecção de Pontos Chave

Para realizar a detecção dos pontos chave, o algoritmo SURF utiliza uma versão simplificada da Matriz Hessiana, baseada na imagem integral para reduzir o custo computacional, denominada então de detecção "Hessiana Rápida" (*Fast-Hessian Detector*) (BAY; TUYTELAARS; GOOL, 2006).

Em uma imagem integral $I_{\sum}(x)$ (também referenciada como *Summed Area Table*), qualquer região localizada em $x = (x, y)$ representa a soma de todos os pixeis da imagem de entrada I em uma região retangular formada pelo ponto x e pela origem, de forma que $I_{\sum}(X) = \sum_{i=0}^{i<x} \sum_{j=0}^{j<y} I(i, j)$ (BAY; TUYTELAARS; GOOL, 2006).

Conforme ilustra a figura 2.9, é possível calcular a soma das intensidades de qualquer área ABCD, independente de seu tamanho, através de $\sum = A - B - C + D$. Assim, o tempo de processamento independe do tamanho da área, essencial para o algoritmo SURF, que utiliza filtros de grandes tamanhos.

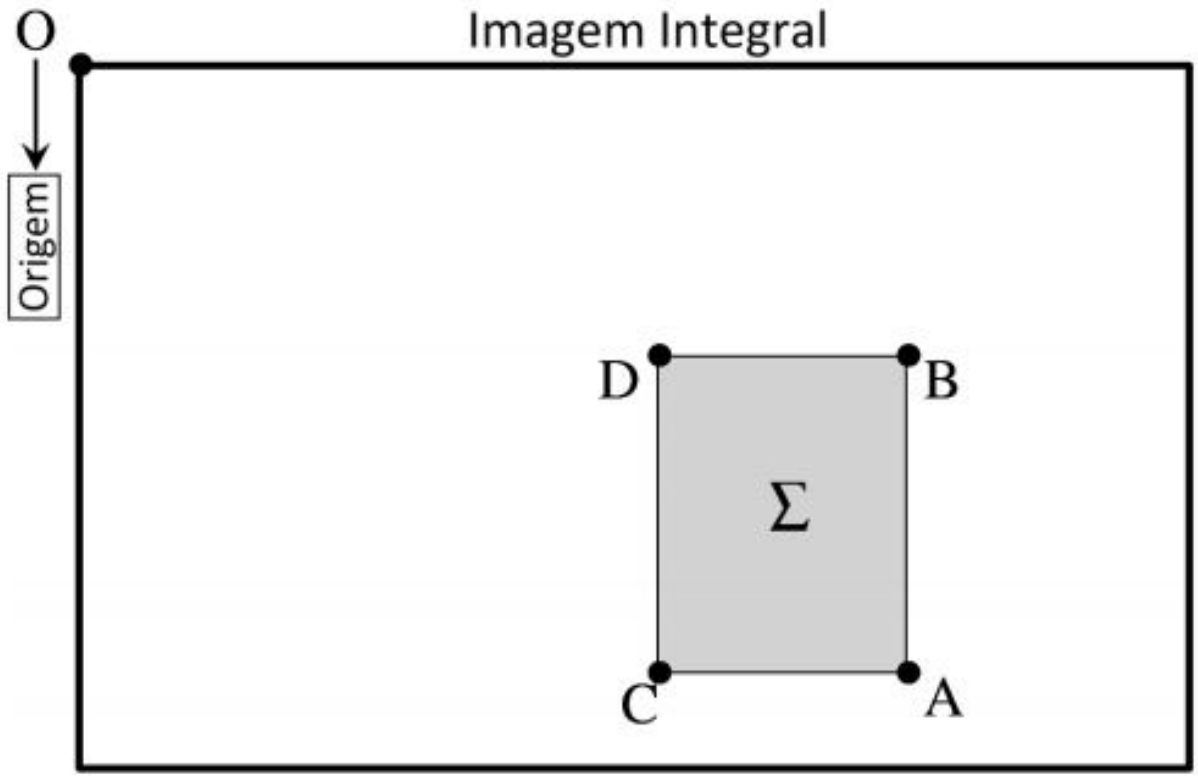

Figura 2.9 – Soma das intensidades de uma região retangular em uma imagem integral. Fonte: (NOVAIS, 2016).

A detecção é baseada na matriz Hessiana devido à sua boa performance e precisão, utilizando o determinante da matriz Hessiana para medir localização e escala

(BAY; TUYTELAARS; GOOL, 2006). Assim, dado um ponto $p_x = (x, y)$ em uma imagem I, a matriz Hessiana $H(x, \sigma)$ em x na escala σ é definida por

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix}$$

onde $L_{xx}(x, \sigma)$, $L_{xy}(x, \sigma)$ e $L_{yy}(x, \sigma)$ representam a convolução da derivada de segunda ordem da função Gaussiana $\frac{\partial^2}{\partial x^2} g(\sigma)$ com a imagem I no ponto x (BAY; TUYTELAARS; GOOL, 2006).

Para aumento de desempenho, (NOVAIS, 2016) o algoritmo faz uso de aproximações das derivadas de segunda ordem da função Gaussiana, presentes na matriz Hessiana. A Figura 2.10 mostra os filtros de caixa baseados nas derivadas de segunda ordem da função Gaussiana.

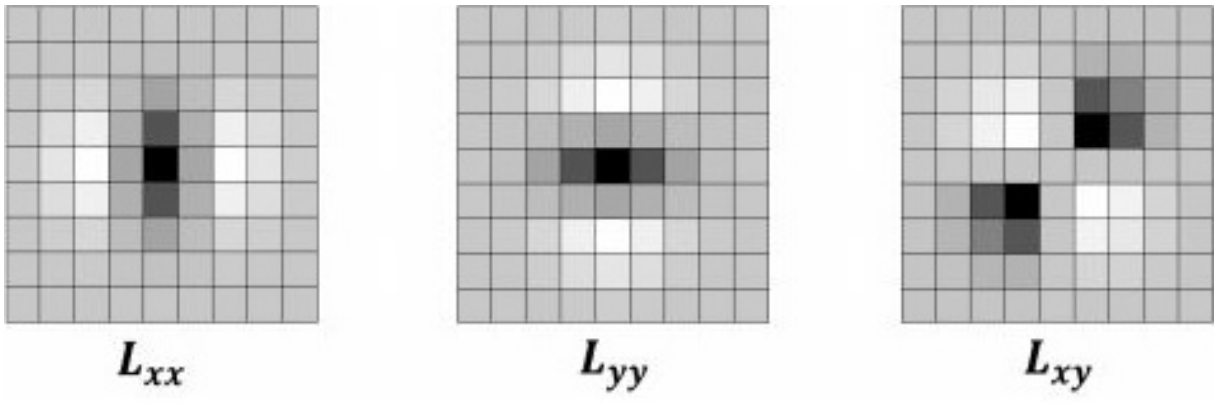

Figura 2.10 – Filtros baseados nas derivadas de segunda ordem da função Gaussiana. Adaptado de: (BAY; TUYTELAARS; GOOL, 2006).

A figura 2.11, por sua vez, mostra os filtros de caixa baseados nas aproximações das derivadas de segunda ordem da função Gaussiana, nestas figuras, as regiões de cor cinza representam o valor 0, regiões claras representam um valor positivo, e regiões escuras um valor negativo.

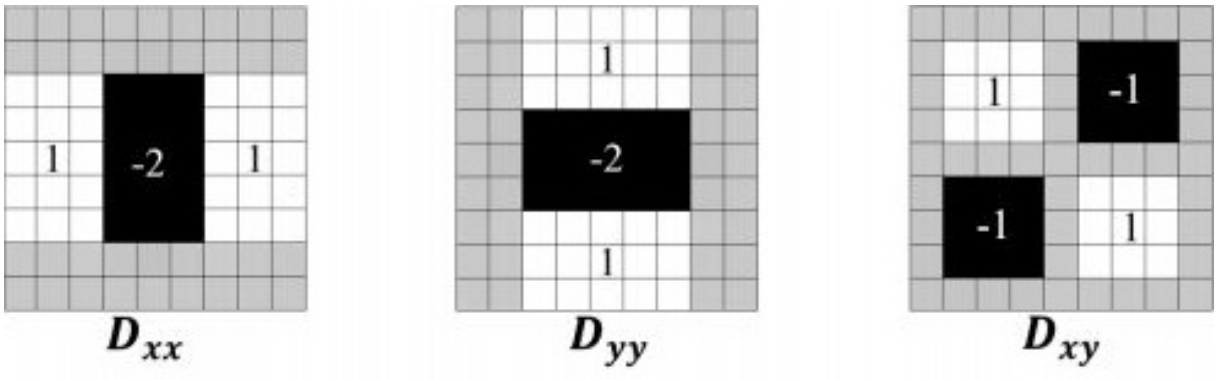

Figura 2.11 – Aproximações das derivadas de segunda ordem da função Gaussiana. . Adaptado de: (BAY; TUYTELAARS; GOOL, 2006).

Os filtros ilustrados na figura 2.11 são aproximações baseadas em $\sigma = 1,2$, que representa a menor escala para computar os mapas de respostas dos cumes (NOVAIS, 2016). Estas aproximações de baixo custo computacional (BAY; TUYTELAARS; GOOL, 2006) são descritas por D_{xx}, D_{xy} e D_{yy} onde o determinante da matriz Hessiana aproximada é dado por

$$det(H_{aprox}) = D_{xx}D_{yy} - (wD_{xy})^2$$

onde $w = 0,912... \simeq 0,9$ é um fator constante utilizado para balancear o determinante da matriz.

O determinante aproximado da matriz Hessiana representa a resposta de um cume na localização x, sendo um possível ponto-chave. Cada resposta é então armazenada em uma mapa de respostas sobre diferentes escalas. Assim, os pontos detectados são procurados através das diferentes escalas, para obtenção apenas dos pontos invariantes à escala (NOVAIS, 2016).

Por utilizar filtro de caixas e imagens integrais, o algoritmo SURF necessita somente aplicar, através de convolução, os filtros de caixa de vários tamanhos diretamente na imagem original, assim, o espaço de escalas é gerado através do aumento da escala do tamanho do filtro de caixa (Figura 2.11) que será convolucionado com a imagem. O resultado da convolução da imagem original com os filtros mostrados na figura 2.10 é considerada como a camada inicial das escalas. As camadas subsequentes são obtidas filtrando esta camada inicial gradualmente com filtros maiores, levando em consideração o uso das imagens integrais e a estrutura específica dos filtros de caixa na figura 2.10.

O espaço de escalas é dividido em oito partes, chamadas de oitavas. Cada oitava é formada por múltiplas respostas obtidas pela convolução da imagem de entrada com filtros de caixa de tamanho crescente. A construção do espaço de escalas começa na primeira oitava, que é formada pela aplicação dos filtros de 9x9, 15x15, 21x21 e 27x27, onde, partindo do filtro 9x9, gera-se filtros maiores com um incremento de 6. Para cada nova oitava, o valor do incremento dos filtros é dobrado, assim, os filtros da segunda oitava são 15x15, 27x27, 39x39 e 51x51 e assim, sucessivamente (NOVAIS, 2016).

Para localizar os pontos chave na imagem utilizando o espaço de escalas, é aplicada uma supressão dos determinantes não-maximos (*non-maximum suppresion*) em uma vizinhança tridimensional de 3x3x3, conforme ilustra a figura 2.12, fazendo com que a supressão ocorra não apenas na vizinhança da imagem, mas também nas escalas vizinhas acima e abaixo (BAY; TUYTELAARS; GOOL, 2006).

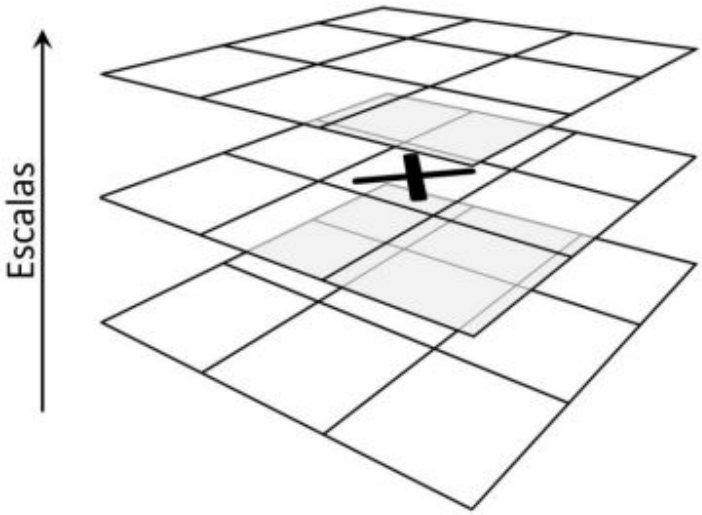

Figura 2.12 – Supressão de não-máximos. Fonte: (NOVAIS, 2016).

Esta supressão é realizada por todo o espaço de escalas de forma que muitos determinantes calculados da matriz Hessiana sejam eliminados, restando apenas os determinantes máximos. Assim, um determinante máximo é selecionado como ponto chave, suprimindo todos os seus vizinhos (NOVAIS, 2016).

2.2.1.2 Descrição de Pontos Chave

Para descrever os pontos chave de maneira invariante à rotação, o algoritmo realiza em primeira instância a identificação da orientação dos pontos chaves. Para isso, calcula-se as respostas do filtros das *wavelets* de Haar nas direções *x* e *y* dentro de uma vizinhança circular cujo raio é dado por *6s* onde *s* é a escala na qual aquele ponto foi detectado, realizando essa operação na imagem integral devido ao seu desempenho de busca(BAY; TUYTELAARS; GOOL, 2006). A figura 2.13 ilustra os filtros *wavelet* de Haar.

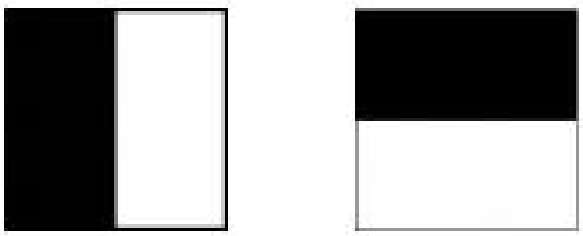

Figura 2.13 – Filtros *wavelet* de Haar utilizados para calcular as respostas nas direções x (à esquerda) e y (á direita). A parte preta tem peso -1 enquanto a parte branca tem peso 1. Fonte: (BAY; TUYTELAARS; GOOL, 2006).

Com as respostas das *waveletes* calculadas e suavizadas com um filtro gaussi-

ano ($\sigma = 2$) centralizado na localização do ponto chave, as respostas são representadas como pontos em um espaço com valor de resposta horizontal na abscissa (eixo x) e valor de resposta vertical na ordenada (eixo y). A orientação dominante é então estimada calculando-se a soma de todas as respostas dentro de uma janela deslizante de orientação com tamanho igual a $\frac{\pi}{3}$, representada pela área cinza na figura 2.14. Ambas as respostas *x* e *y* somadas produzem o valor do vetor de orientação local para aquela janela. A orientação do ponto de interesse será definida então pelo vetor que, dentre todas as janelas deslizantes, possuir maior tamanho.

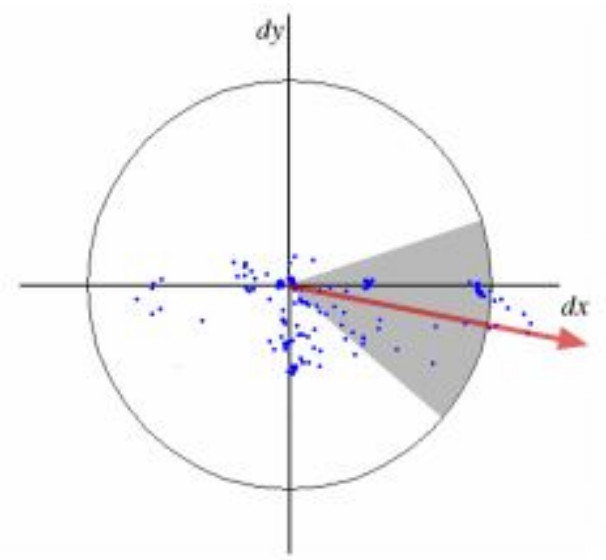

Figura 2.14 – Definição de Orientação. Uma janela deslizante de orientação com tamanho $\frac{\pi}{3}$, utilizada para detectar a orientação das respostas. Fonte: (BAY; TUYTELAARS; GOOL, 2006).

Uma vez que a orientação dos pontos chave foi definida, constrói-se então uma região quadrada, centrada ao redor do ponto chave, de tamanho $20s$ onde *s* a escala daquele ponto e orientada de acordo com a orientação definida no passo anterior (BAY; TUYTELAARS; GOOL, 2006). Em seguida, a região é dividida em uma matriz 4x4 de sub-regiões menores, preservando assim informação espacial importante. Para cada uma dessas sub-regiões, calcula-se as repostas da *wavelet* em pontos regularmente espaçados para uma área de 5x5.

A resposta da *wavelet* de Haar na horizontal é chamada de *dx*, e a resposta da *wavelet* de Haar na vertical, de *dy*, para um filtro de tamanho *2s*. Neste ponto, “horizontal” e “vertical” são definidos em relação à orientação do ponto de interesse selecionado, conforme ilustra a figura 2.15. Para melhorar os resultados em relação à deformações geométricas e erros de localização, a resposta *dx* e *dy* são suavizadas com um filtro Gaussiano ($\sigma = 3.3s$) no centro do ponto.

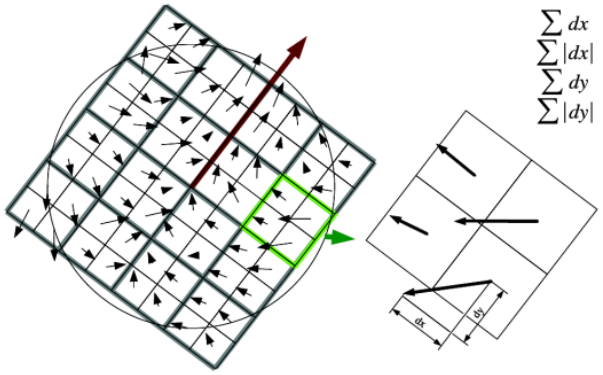

Figura 2.15 – Para construir o descritor, uma região quadrada e orientada dividida em um grade com 4x4 sub regiões é impressa sobre o ponto de interesse (à esquerda). Para cada quadrado, a resposta do *wavelet* é calculada. As subdivisões 2x2 de cada quadrado corresponde aos campos atuais do descritor. Estas são as somas de *dx*, |*dx*|, *dy* e |*dy*|, calculadas relativas à orientação da grade (à direita).. Traduzido de: (BAY; TUYTELAARS; GOOL, 2006).

Para cada subregião, as respostas dos waveletes em *dx* e *dy* são somadas e formam o primeiro conjunto de entradas do vetor descritor. O segundo conjunto de entradas do vetor descritor é formado pela soma dos módulos das respostas dos wavelets (|*dx*| e |*dy*|). Dessa maneira, o vetor descritor *v* para o ponto chave é representado por $v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$. Assim, ao realizar a operação para cada uma das 16 subregiões, obtemos um vetor descritor com 64 valores.

2.2.2 SIFT

Lowe (1999) propôs o algoritmo SIFT (*Scale Invariant Features Transform*), capaz de detectar e descrever pontos chave, atribuindo um vetor descritor para cada ponto. Tal qual o algoritmo SURF, o algoritmo SIFT trabalha apenas com o nível de cinza da imagem em suas etapas.

As etapas do agoritmo SIFT serão abordadas a seguir.

2.2.2.1 Detecção de Pontos Chave

Para detecção dos pontos chave, (LOWE, 2004) o algoritmo SIFT primeiramente procura por potenciais pontos chave, analisando a imagem em múltiplas escalas possíveis, utilizando para isso, o espaço de escalas. para construir o espaço de escalas para uma imagem de entrada $I(x,y)$ é necessário aplicar um filtro Gaussiano $G(x,y,\sigma)$ sobre a imagem, (onde σ representa a escala), definido por

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Assim, o espaço de escalas $L(x,y,\sigma)$ é constituído por

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

onde * denota a convolução em ***x*** e ***y***.

O espaço de escalas é dividido em oitavas, dependendo da aplicação. Em cada oitava, o filtro Gaussiano é aplicado múltiplas vezes na imagem para produzir o conjunto $L(x,y,\sigma)$, separadas pelo valor de ***k***, conforme a parte esquerda (Filtro Gaussiano) da imagem 2.16. Lowe (1999) recomenda como bons valores, 4 oitavas, 5 níveis de escala, $\sigma = 1,6$ e $k = \sqrt{2}$.

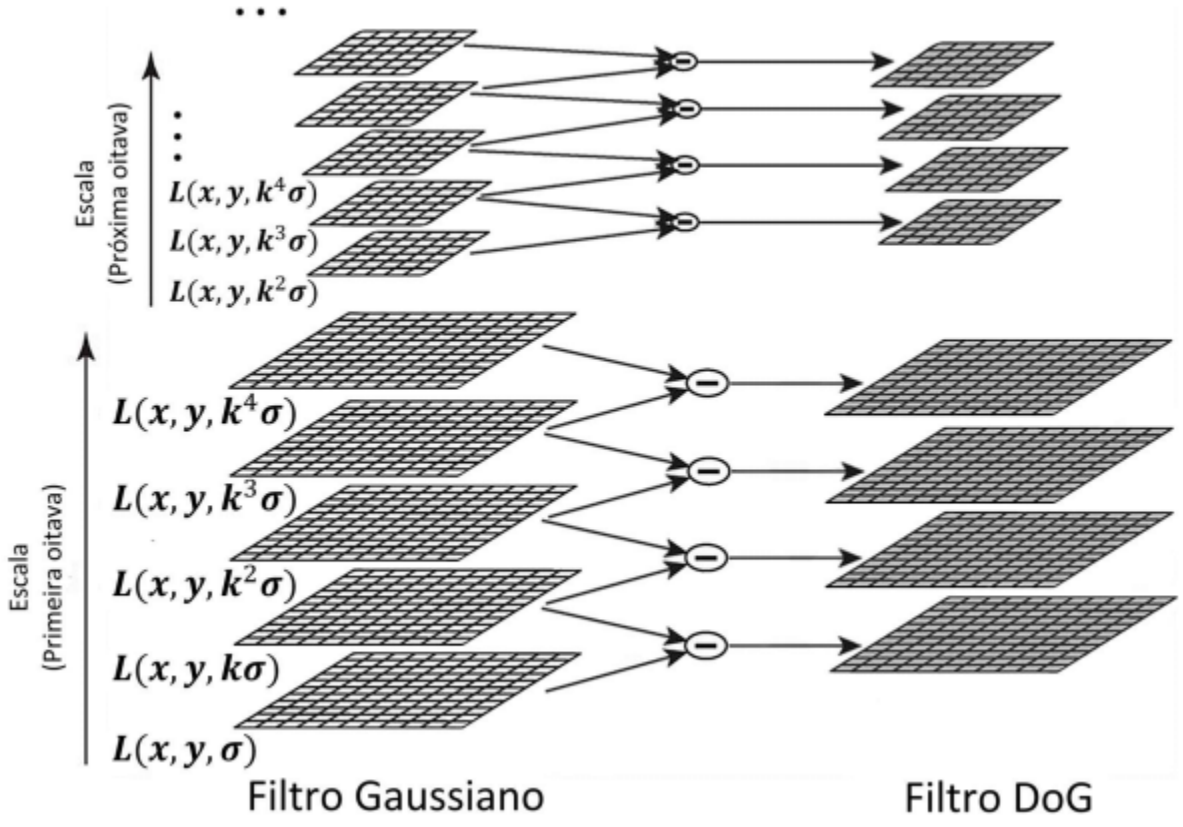

Figura 2.16 – Construção do espaço de escala. Fonte: (NOVAIS, 2016).

Após gerar o os conjuntos $L(x,y,k^n\sigma)$ do espaço de escala, o algoritmo calcula as imagens denominadas $D(x,y,\sigma)$ através da técnica de diferença de gaussianos (*Diference of Gaussians*, ou apenas *DoG*). O filtro DoG é uma aproximação menos custosa do filtro LoG (*Laplacians of Gaussians*), e serve para encontrar localizações estáveis dos pontos chave no espaço de escala. Uma imagem $D(x,y,\sigma)$ é obtida através da subtração de duas imagens da escala, definida por

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma)$$

conforme pode ser vista na parte direita (Filtro DoG) da figura 2.16.

Após processar as oitavas, a imagem com filtro Gaussiano aplicado cujo valor de σ seja o dobro do valor inicial de σ tem a sua resolução diminuída pela metade, e todo o processo é repetido a partir da mesma. Esse processo resulta em uma pirâmide de escalas.

Uma vez calculados os DoG, as imagens são varridas em busca dos extremos locais no espaço de escalas, analisando os oito pixeis vizinhos na escala atual, assim como seus nove vizinhos na escala anterior, e seus nove vizinhos na próxima escala, conforme ilustra a figura 2.17, onde o x representa o pixel que está sendo analisado.

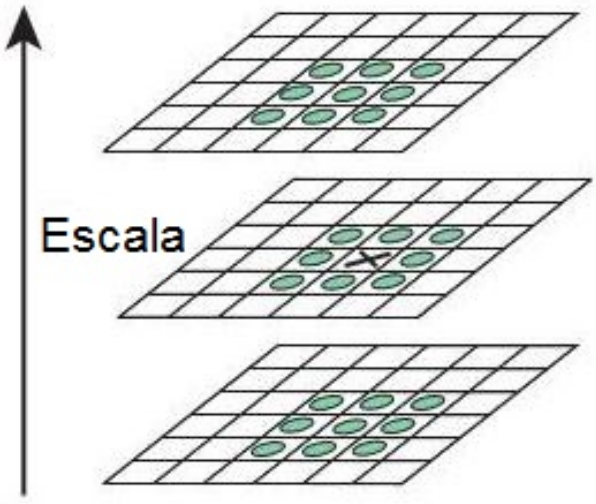

Figura 2.17 – Busca dos extremos locais no espaço de escalas. Traduzido de: (LOWE, 2004).

Caso o pixel seja um extremo local (possui maior ou menor valor dentre todos os 27 pixeis vizinhos), o mesmo é considerado um potencial ponto chave.

Após a obtenção dos potenciais pontos chave, é necessário refinar os resultados para obtenção de resultados mais precisos, descartando os pontos potenciais gerados por ruídos ou que não sejam considerandos bons.

Assim, realiza-se a determinação da localização interpolada dos extremos, utilizando a técnica de expansão de Taylor no espaço de escala das imagens $D(x,y,\sigma)$ onde neste espaço é realizado um deslocamento para que a origem da expansão seja definida no potencial ponto chave de amostra (NOVAIS, 2016). A expansão de Taylor é denotada por

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{2.4}$$

onde D é o valor de $D(x,y,\sigma)$ no ponto de amostra, $x = (x,y,\sigma)^T$ é o vetor de deslocamento que parte do ponto de amostra e $D(x)$ é uma aproximação do valor

interpolado de $D(x,y,\sigma)$ para um ponto translado pelo deslocamento x.

Para determinar a localização do extremo, representado por x', é calculada a derivada da equação 2.4 em relação à x, igualando seu resultado à 0, sendo então

$$x' = -\frac{\partial^2 D}{\partial x^2}^{-1} \frac{\partial D}{\partial x}$$

e, caso $x' > 0.5$ em qualquer uma de suas dimensões, o extremo se encontra mais próximo de outro ponto de amostra. Nestes casos, o ponto de amostra é alterado através da adição de x' à sua localização, porém, sua localização interpolada deverá ser recalculada para que a localização estimada do extremo seja obtida.

A função $D(x')$ definida por

$$D(x') = D + \frac{1}{2}\frac{\partial D^T}{\partial x} x1$$

tem como resultado a intensidade de um ponto, sendo utilizada para descartar os potenciais pontos chave cujo valor de contraste esteja abaixo de um limiar $ct = 0.03$ (LOWE, 2004).

Além de pontos de baixo contraste, (NOVAIS, 2016) o algoritmo também descarta potenciais pontos chaves que estejam mal localizados ao longo de arestas. A aplicação do filtro de Gaussiano gera potenciais pontos chave mal localizados cuja magnitude de curvatura principal será maior ao longo da aresta, porém, pequena na direção perpendicular. Para calcular essa magnitude, é utilizada uma matriz Hessiana H, calculada na mesma localização e escala do potencial ponto chave analisado, sendo definida por

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Para encontrar os valores de D_{xx}, D_{xy} e D_{yy}, calcula-se então as derivadas dessa matriz através da aproximação de diferenças entre pontos vizinhos do ponto que está sendo analisado, (GONZALES; WOODS, 2011) de forma que podemos encontrar então

$$D_{xx} = D(x+1,y,\sigma) - 2D(x,y,\sigma) + D(x-1,y,\sigma)$$

$$D_{xy} = \frac{D(x-1,y+1,\sigma) - D(x+y,y+1,\sigma) + D(x+y,y-1,\sigma) - D(x-y,y-1,\sigma)}{4}$$

$$D_{yy} = D(x,y+1,\sigma) - 2D(x,y,\sigma) + D(x,y-1,\sigma)$$

Os autovalores de H são proporcionais as curvaturas principais, tornando possível mensurar através deles, a magnitude das curvaturas principais. Considerando α o autovalor de maior magnitude e β o de menor magnitude, a soma dos valores é

calculada pelo traço de H e o produto dos autovalores é calculados pelo determinante de H:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

As curvaturas possuem sinais diferentes quando $Det(H) < 0$ e, portanto, o ponto examinado é descartado.

Sendo r a relação entre os autovalores α e β de maneira que $\alpha = r\beta$, conclui-se que

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

Desta maneira, todos os potenciais pontos chave que não satisfaçam

$$\frac{Tr(H)^2}{Det(H) < \frac{(r+1)^2}{r}}$$

são descartados. Desta forma, pontos chaves que tenham uma razão entre as curvaturas principais maior que r (Lowe (2004) recomenda $r = 10$) são eliminados.

2.2.2.2 Descrição de Pontos Chave

O primeiro passo do algoritmo SIFT para descrever os pontos chave consiste em determinar uma orientação consistente cada cada ponto chave com base nas propriedades locais da imagem. Dessa maneira, o descritor do ponto chave não será influenciado por rotação, garantindo que os pontos chave sejam identificados mesmo que a imagem seja rotacionada.

Para isto, utiliza-se a escala do ponto chave para selecionar uma imagem L, suavizada pelo filtro Gaussiano que esteja em uma escala mais próxima, de forma que todos os cálculos realizados também sejam invariantes à rotação.

Desta forma, para cada ponto $L(x,y)$ em L, realiza-se o pré-calculo do gradiente $\theta(x,y)$ e da magnitude $m(x,y)$, com base nas diferenças dos pixeis. O gradiente e a magnitude podem ser definidos por

$$\theta(x,y) = tan^{-1}(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)})$$

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

Em seguida, um histograma de orientações com 36 campos (que cobre 360º) é criado a partir das orientações da magnitude e gradiente dos vizinhos ao redor do ponto chave. Cada vizinho adicionado ao histograma é ponderado através de sua

magnitude do gradiente assim como sua magnitude em uma janela circular com filtro Gaussiano, onde σ equivale à 1,5 vezes a escala do ponto chave (NOVAIS, 2016).

Os valores armazenados no histograma correspondem as direções dominantes dos gradientes locais. O campo do histograma com maior valor é selecionado e utilizado para analisar os outros campos. Para cada outro campo, verifica-se se o mesmo possui ao menos 80% do valor do maior campo. Os campos que satisfaçam essa condição serão utilizados, criando novos pontos chave na mesma localização e escala, porém, com a direção do histograma. Desta maneira, locais com muitos picos de magnitude similares terão múltiplos pontos chave na mesma localização e escala, mas com diferentes orientações. Isso contribui para a estabilidade das operações de correspondência.

A figura 2.18 ilustra a obtenção dos descritores através do algoritmo SIFT.

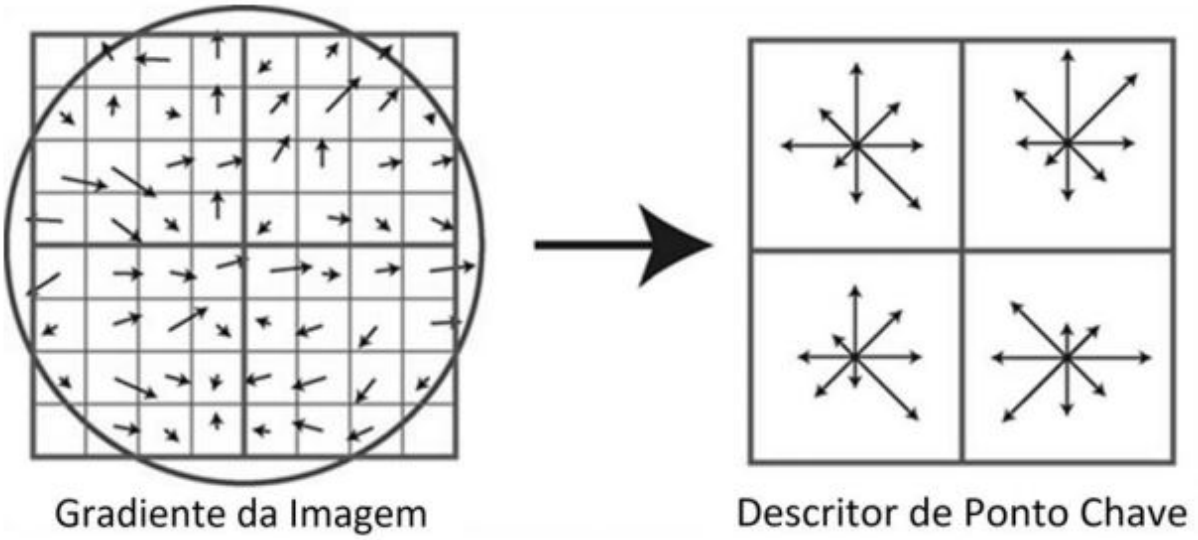

Figura 2.18 – Processo de descrição dos pontos chave no algoritmo SIFT. Fonte: (NOVAIS, 2016).

Neste ponto, os pontos chaves já foram definidos e os gradientes já foram previamente calculados para todos os níveis na pirâmide de escala no passo anterior. Então, para atingir a invariância à rotação, as coordenadas do descritor e das orientações do gradiente são rotacionadas em relação à orientação do ponto chave (LOWE, 2004).

O lado direito da figura 2.18 ilustra um descritor de ponto chave onde os quatro histogramas são formados pela soma das orientações e magnitudes do gradiente da sua respectiva sub região, ilustradas na esquerda da figura 2.18.

Então, pode-se afirmar que o vetor descritor no algoritmo SIFT é formado por um vetor contendo os valores de todas as entradas dos histogramas de orientação.

A figura 2.18 ilustra um vetor de histograma de orientação de 2x2, com 8 campos de orientação cada. No algoritmo SIFT, são utilizados vetores descritores 4x4, com 8 campos cada, o que resulta em vetor descritor de características com 128 elementos para cada ponto chave.

O vetor obtido então é modificado de forma a reduzir os efeitos de alterações na iluminação, sendo normalizado em uma unidade de comprimento. Depois , os valores do vetor são limitados de forma que não sejam maiores que 0,2 e, então, o vetor é novamente normalizado em uma unidade de comprimento (NOVAIS, 2016).

2.2.3 ORB

2.2.3.1 Detecção de Pontos Chave

O algoritmo ORB utiliza para detecção dos pontos chave, o algoritmo FAST, abordado na subseção 2.1.2, entretanto, Rublee et al. (2011) realizou alterações no comportamento do algoritmo FAST de maneira que cada ponto chave detectado possuísse informação sobre a sua orientação. Essa alteração foi denominada de FAST Orientado, ou apenas *oFAST* (*Oriented FAST*).

Dessa maneira, o primeiro passo realizado pelo algoritmo é a obtenção de pontos chave FAST, utilizando um circulo de raio 9 ao redor de um píxel p, o que mantém boa performance.

O algoritmo FAST não produz uma medida de angularidade do canto (*cornerness*) e gera uma grande quantidade de falsos positivos em arestas (RUBLEE et al., 2011). É utilizado então o medidor de angularidade proposto por Harris e Stephens (1988) para ordenar os pontos chaves pela sua angularidade. Sendo N a quantidade de pontos chave desejados, é definido um limar baixo o suficiente para obter mais que N pontos chaves e, ordenando-os com o medidor de Harris e Stephens (1988) e selecionando apenas os N primeiros.

O algoritmo FAST também não trabalha com pontos chave em escala (RUBLEE et al., 2011). Para isso, é criado uma pirâmide de espaço escala da imagem original e detectados os pontos chave FAST (também filtrados através do medidor de Harris e Stephens (1988)) em cada nível da pirâmide.

Para descobrir a orientação dos cantos detectados, foi utilizada uma técnica de medição de orientação de cantos chamada centroide de intensidade (*intensity centroid*)(ROSIN, 1999). O centroide de intensidade considera que a intensidade de um canto estará deslocada de seu centro, e a orientação do canto pode ser definida por um vetor com origem no centro até o centroide. Conforme definido por Rosin (1999),

esse vetor pode ser encontrado utilizando

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

como definição dos "*moments of a patch*". Assim, calculados "*the moments*", podemos calcular o centroide a partir de

$$C = \left(\frac{m_1 0}{m_0 0} \frac{m_0 1}{m_0 0} \right)$$

Assim, podemos calcular o vetor $\vec{OC}$ com origem no centro do canto, denotado por O, e extremidade no centroide, denotado por C. A orientação do "patch" então pode ser encontrada por $\theta = atan2(m_0 1, m_1 0)$, onde atan2 denota um arco tangente (ângulo entre o eixo x e uma linha que contém a origem $(0,0)$ e um ponto com coordenadas (x,y)).

Para melhorar a invariância à rotação destas medidas, os "*moments*" são calculados com um x e y dentro da região circular do raio r. Rublee et al. (2011) define empiricamente que r tenha o mesmo tamanho do "*patch size*", de maneira que x e y "*run from*" $[-r,r]$. Quando $|C|$ se aproxima de 0, a medida se torna instável, porém, a ocorrência de $|C| \approx 0$ é rara.

Rublee et al. (2011) comparou o método baseado em centroide com duas medidas baseadas no gradiente, BIN e MAX e afirmou que ambos BIN e MAX não obtiveram bons desempenhos, enquanto o método baseado em centroide calcula a orientação de forma uniforme, mesmo em ambientes com alto nivel de ruído. A figura 2.19 demonstra as variações ao serem executados em um ambiente de testes controlado com rotação no plano e adição de ruído.

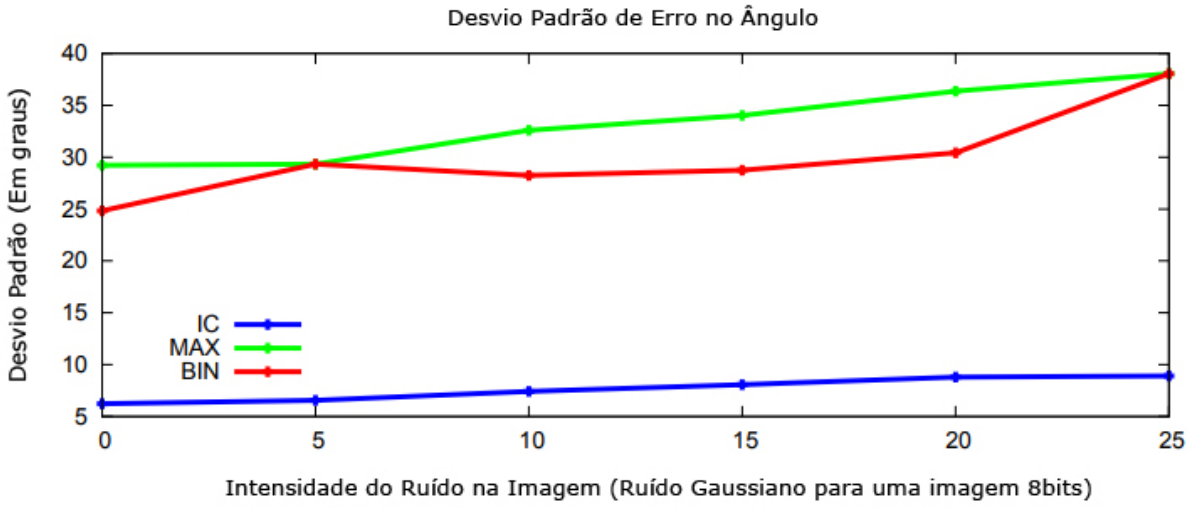

Figura 2.19 – . Medição de Desempenho Na Rotação. Centroide de Intensidade (IC) possui melhor desempenho comparado com BIN e MAX. Traduzido de: (RUBLEE et al., 2011).

2.2.3.2 Descrição de Pontos Chave

Para descrever os pontos chave, o ORB utiliza uma variação dirigida do algoritmo BRIEF (CALONDER et al., 2010), chamada de rBRIEF (*Rotation-Aware Brief*, ou "Brief Ciente de Rotação"), uma vez que o algoritmo BRIEF não produz bons resultados quando ocorre rotações. Assim, o ORB redireciona o BRIEF de acordo com a orientação dos pontos de interesse (RUBLEE et al., 2011).

O descritor BRIEF consiste em descrever em um vetor de bits uma pequena região da imagem através de conjunto de testes binários de intensidade. Considerando uma região p da imagem, suavizada, o teste binário τ pode ser definido por

$$\tau(p;x,y) := \begin{cases} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{cases}$$

onde $p(x)$ é a intensidade de p na posição x. O ponto chave então será definido por um vetor de n testes binários

$$f_n(p) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(p;x_i,y_i)$$

sendo $n = 256$ para o algoritmo ORB. Antes de executar os testes, é de suma importância suavizar a imagem. O algoritmo ORB suaviza através de uma imagem integral, onde cada caso de teste é uma janela de 5x5 em uma região de 31x31 pixeis.

Para que então os descritores BRIEF se tornem invariantes à rotação no plano, o algoritmo ORB realiza o redirecionamento do descritor de acordo com a orientação do ponto chave. Assim, para qualquer conjunto de pontos chave de n testes binários em uma localização (x_i, y_i), define-se a matriz *S* de tamanho 2 X n

$$S = \begin{pmatrix} x_1, ..., x_n \\ y_1, ..., y_n \end{pmatrix}$$

Usando a direção θ da região e a matriz de rotação R_θ, direciona-se S de maneira a construir

$$S_\theta = R_\theta S$$

Dessa maneira, o operador do BRIEF dirigido será

$$g_n(p, \theta) := f_n(p) | (x_i, y_i) \in S_\theta$$

O angulo então é discretizado em incrementos de $\frac{2\pi}{30}$ (12º) de maneira a montar uma tabela de busca de padrões BRIEF pré-computados. Assim, a orientação θ do ponto chave sendo consistente pelas janelas, o conjunto S_θ será utilizado para calcular seu descritor (RUBLEE et al., 2011).

2.3 Algoritmos de Correspondência

Para encontrar pontos chave correspondentes em diferentes imagens, é necessário utilizar os descritores abordados no capítulo anterior.

A correspondência utiliza diferentes técnicas para comparar descritores em duas imagens, e encontrar qual a melhor correspondência entre ambas. O conceito de correspondência utiliza como base o cálculo de distância entre descritores.

A seguir, serão abordados os conceitos matemáticos de distância e técnicas de correspondência de pontos chave.

2.4 Distâncias

Na área computacional, a distância entre duas palavras-código está relacionada à quantidade de operações necessárias para ir de uma palavra-código A para uma palavra-código B (TANEMBAUM, 1990).

Esta seção aborda duas distâncias importantes para visão computacional: A distância Euclidiana e a distância de Hamming.

2.4.1 Distância Euclidiana

Distância Euclidiana (também referenciada como distância métrica) é a distância entre dois pontos. A distância Euclidiana pode ser provada através de múltiplas aplicações do teorema de Pitágoras (HOWARD, 2001).

A distância Euclidiana pode ser computada de maneiras diferentes, dependendo do escopo em que a mesma está sendo aplicada. Na visão computacional, mais especificamente na correspondência de características, é realizada a análise de distância entre vetores descritores.

Assim, para dois conjuntos de descritores, $P = (p_1, p_2, ..., p_n)$ e $Q = (q_1, q_2, ..., q_n)$, a distância entre um ponto p_i e q_i dependerá da dimensão do vetor analisado.

Caso o vetor descritor seja unidimensional, (HOWARD, 2001) a distância Euclidiana será medida através de

$$\sqrt{(p_x - q_x)^2} = |p_x - q_x|$$

onde $P = (p_x)$ e $Q = (q_x)$. Caso o vetor seja bidimensional onde $P = (p_x, p_y)$ e $Q = (q_x, q_y)$, (HOWARD, 2001) o vetor é calculado por

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Dessa maneira, é possível afirmar que para um conjunto *n*-dimensional $P = (p_x, p_y, p_z, ..., p_i)$ e $Q = (q_x, q_y, q_z, ..., q_i)$ o cálculo da distância Euclidiana é dado por

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2 + ... + (p_i - q_i)^2}$$

2.4.2 Distância de Hamming

A distância de Hamming surgiu como uma maneira de identificar e corrigir erros em palavras binárias (TANEMBAUM, 1990). Dadas duas palavras quaisquer, é possível determinar quantas são as diferenças entre as duas.

Considerando 10001001 e 10110001 como duas palvras binárias, encontra-se através da operação XOR (OU Exclusivo) entre ambas quantos bits diferem verificando-se os bits 1. Assim, (sendo $\oplus$ o simbolo de XOR)

$$100001001 \oplus 101100001 = 00111000$$

possui três bits 1, ou seja, diferem em três bits. O número de posições nas quais duas palavras diferem é chamado de distância de Hamming (TANEMBAUM, 1990).

A distância de Hamming não se limita apenas à palavras binárias, podendo ser aplicadas em strings

$$elabore \oplus melhore = 1111000 = 4$$

ou em matrizes

$$\begin{bmatrix} 128 & 316 \\ 415 & 256 \end{bmatrix} \oplus \begin{bmatrix} 128 & 316 \\ 407 & 256 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = 1$$

Assim, a distância de Hamming é também utilizada para verificar diferenças em vetores descritores, verificando quantos valores diferem entre cada conjunto de dados.

2.5 Técnicas de Busca e Correspondência

Para buscar um ponto existente em um conjunto em outro conjunto e realizar a correspondência entre os mesmos, é necessário realizar uma busca do ponto $p_n \in P$ em Q, onde p_n é um ponto qualquer no conjunto P que pode ou não estar contido no conjunto Q (RUBLEE et al., 2011).

Esta seção aborda as técnicas de busca e correspondência de pontos chave entre dois conjuntos que serão avaliadas neste trabalho.

2.5.1 *BFM - Brute Force Matcher*

A técnica de correspondência de pontos chave por força bruta é uma técnica simples, mas computacionalmente custosa (GONZALES; WOODS, 2011).

A técnica de força bruta utiliza uma medida de distância (a implementação em OpenCV utiliza por padrão a distância Euclidiana) para realizar a comparação. Para isso, compara-se cada descritor de um conjunto de descritores com todos os outros descritores de um segundo conjunto. O descritor do segundo conjunto que possuir a menor distância é retornado como correspondente.

Algumas implementações da correspondência por força bruta realiza dupla verificação, assim, para um ponto chave p do primeiro conjunto que obteve p' do segundo conjunto como correspondente, verifica-se também se para p', p é o melhor correspondente (GONZALES; WOODS, 2011).

A dupla verificação garante maior confiabilidade ao algoritmo ao custo de desempenho de processamento.

2.5.2 *KNN - K Nearest Neighbors*

Dado um conjunto de n pontos $P = p_1, p_2, p_3, ..., p_n$ em um espaço métrico M e uma localização espacial $q \in M$, um algoritmo NN (*Nearest Neighbors*) busca um elemento $NN(q,P) \in P$ que esteja o mais próximo de q respeitando a distância métrica $d : MxM \rightarrow \mathbb{R}$ (MUJA; LOWE, 2009) de maneira que

$$NN(q,P) = argmin_{x \in P} d(q,x)$$

O algoritmo *KNN* tem como objetivo retornar k vizinhos mais próximos de q de maneira que

$$KNN(q,P) = A$$

onde A é um conjunto que satisfaça as condições

$$|A| = K, A \subseteq P$$
$$\forall_x \in A, y \in P - A, d(q,x) \leq d(q,y)$$

Para realizar essa busca, o algoritmo *KNN* utiliza a mesma metodologia do algoritmo *BFM*, ordenando os vizinhos pela distância de maneira crescente, porém, ao invés de trazer o mais próximo como correspondente, retorna K vizinhos.

A busca através do algoritmo KNN sempre retornará exatamente K vizinhos (caso houver ao menos K pontos em P) (MUJA; LOWE, 2009).

2.5.3 *FLANN - Fast Library for Approximate Nearest Neighbors*

Muitas pesquisas foram desenvolvidas para resolver o problema de busca de vizinhos próximos, porém, poucas foram as pesquisas conduzidas à respeito da escolha de algoritmos já existentes, assim como sobre a definição de valores para os parâmetros dos mesmos. Assim, a escolha de um algoritmo e a definição de seus parâmetros é, em sua maioria, um processo manual e raramente faz uso de abordagens mais sistemáticas (MUJA; LOWE, 2014).

FLANN (MUJA; LOWE, 2009) é uma biblioteca que consiste em um conjunto de algoritmos otimizados para uma busca ágil em vizinhos próximos própria para espaços de grandes dimensões.

2.5.3.1 Classificação dos Algoritmos NN

O FLANN classifica os algoritmos de busca em três categorias (MUJA; LOWE, 2014): Árvores de Particionamento Binário de Espaço, Técnicas Baseadas em Resumo e Técnicas de Grafos de Vizinhos mais Próximos.

2.5.3.1.1 Árvores de Particionamento Binário de Espaço

Uma árvore k-d (Árvore K-Dimensional) é um dos melhores algoritmos para vizinhos mais próximos conhecido que, apesar de se comportar muito bem em espaços de pequena dimensão, perde desempenho muito rapidamente para dados de grandes dimensões (MUJA; LOWE, 2014).

Arya et al. (1998) propôs uma variação da árvore k-d para realizar buscas aproximadas considerando $(1+\varepsilon)$ vizinhos aproximadamente mais próximos, pontos para os quais $dist(p,q) \leq (1+\varepsilon)dist(p*,q)$ onde $p*$ é o vizinho mais próximo real. Este método também é referenciado como *error bound* (delimitado por erro).

Outra maneira de aproximar a busca pelo vizinho mais próximo é através da limitação do tempo gasto na busca. Este método foi proposto por Beis e Lowe (1997) onde a busca na árvore k-d é interrompida após examinar um número fixo de folhas da árvore. Na prática, (MUJA; LOWE, 2014) buscar utilizando critérios limitadores de tempo se mostrou mais eficiente em retornar resultados melhores que utilizar critérios por erro. Este método também é referenciado como *time bound* (delimitado por tempo).

Em (HARTLEY; SILPA-ANAN, 2008) foi proposto múltiplas árvores k-d randomizadas como tentativa de acelerar o processo de busca dos vizinhos aproximadamente mais próximos. Muja e Lowe (2009) realizou uma gama de comparações demonstrando que o método de múltiplas árvores k-d é o mais efetivo para correspondência em dados de grandes dimensões.

Outras variações da árvore k-d usando hiperplanos de particionamento não alinhados foram propostas: Árvore PCA; Árvore R-T; Árvore de Projeção Binária. Muja e Lowe (2014) afirmam que estes algoritmos não são mais eficientes que uma decomposição da árvore k-d randomizada, uma vez que a sobrecarga da avaliação em múltiplas dimensões durante a busca ultrapassou os benefícios de melhor decomposição espacial.

Ainda existem outras classes de árvores de particionamento que decompõe o espaço usando múltiplos algoritmos de agrupamento ao invés de utilizar hiperplanos como as árvores k-d e suas variantes, como: Árvore Hierárquica *K-Means*; GNAT; Hierarquia Ancorada; Árvore VP; Árvore *Cover* e Árvore Spill. (MUJA; LOWE, 2014) descreveram um algoritmo de árvores *k-means* modificado que obtém melhores resul-

tados para um conjunto específico de dados, enquanto árvores k-d são melhores para outros conjuntos.

2.5.3.1.2 Técnicas Baseadas em Resumo (*Hash Based*)

O método de Resumo Sensível à Localidade, também referenciada por *LSH* (*Local Sensity Hashing*) utiliza múltiplas funções de resumo de maneira que os resumos de elementos próximos também tendem a ser próximas. Variações do *LSH* como LSH de Sonda Múltipla (*Multi-Probe LSH*) melhora o custo de armazenamento diminuindo o número de tabelas de resumo, e o LSH Floresta (*LSH Forest*) se adapta melhor aos dados analisados sem necessidade de ajustes manual dos parâmetros (MUJA; LOWE, 2014).

O desempenho dos métodos baseados em resumo está diretamente relacionado à qualidade das funções utilizadas, assim, várias técnicas cujo objetivo é melhorar os métodos de resumo utilizando funções dependentes dos dados obtidos usando múltiplas técnicas de aprendizados foram desenvolvidas (MUJA; LOWE, 2014): Resumo Sensível à Parâmetros (*Parameter Sensitive Hashing*); Resumo Espectral (*Spectral Hashing*); Resumo LSH Randomizada por Métricas Aprendidas (*Randomized LSH Hasing from Learned Metrics*); LSH Kernelizado (*Kernelized LSH*); Binários Incorporados Aprendidos (*Learnt Binary Embeddings*) entre outros.

Diferentes algoritmos LSH fornecem garantias teóricas na qualidade da busca e vêm sendo utilizados com sucesso em um grande número de projetos, entretanto, os experimentos realizados por Muja e Lowe (2014) demonstram que na prática eles são comumente superados por algoritmos que utilizam estruturas de particionamento espacial como a árvore k-d randomizada e a busca prioritária por árvore *k-means*.

2.5.3.1.3 Técnicas de Grafos de Vizinhos mais Próximos

Os métodos de grafos de vizinho mais próximo criam uma estrutura de grafo em que os pontos são vértices e arestas conectam cada ponto aos vizinhos mais próximos. Os pontos de consulta são usados para explorar este grafo utilizando várias estratégias para se aproximar de seus vizinhos mais próximos.

Em Sebastian e Kimia (2002), os autores selecionam alguns pontos bem separados no grafo e os classificam como sementes, iniciando a busca no grafo nestas sementes buscando o primeiro melhor. Hajebi et al. (2011) realizou uma busca do primeiro melhor de maneira similar, porém utilizou uma estratégia de escalada selecionando os pontos iniciais de maneira aleatória. Experimentos recentes indicam que estes dois métodos se comportam favoravelmente comparados com árvores k-d randomizadas.

Os métodos baseados em grafos de vizinhos mais próximos possuem a desvantagem do custo de construção da estrutura do grafo k-NN.

2.5.3.2 Configuração Automática do Algoritmo NN

O desempenho do algoritmo *LSH* é diretamente ligado ao tamanho do comprimento da chave de resumo. Assim, o algoritmo *LSH Forest* (BAWA; CONDIES; GANESAN, 2005) que realiza um auto-ajuste, eliminando este parâmetro dependente dos dados, foi proposto.

Muja e Lowe (2014) propôs em um trabalho anterior (MUJA; LOWE, 2009) um método para configuração automática de algoritmos NN, combinando busca em grade com o processo de otimização (*simplex*) de Nelder e Mead (1965).

Várias pesquisas sobre métodos de configuração de algoritmos foram realizadas, no entanto, Muja e Lowe (2014) afirmam que nenhuma delas aplicam técnicas para encontrar parâmetros ótimos para algoritmos NN.

Bergstra e Bengio (2012) afirmam que, exceto para lugares com poucos parâmetros, a busca aleatória é uma estratégia mais eficiente para a otimização de parâmetros do que a busca em grade.

2.6 Considerações finais

Conforme analisado em Patel et al. (2014) e (RUBLEE et al., 2011), em aplicações de processamento em tempo real, o algoritmo ORB se destaca dentre os algoritmos SURF, SIFT, FREAK e BRISK, sendo equiparado apenas pelo algoritmo BRIEF, que apesar de também possuir bom desempenho, não se comporta tão bem quanto o ORB em ambientes de baixa luminosidade.

Como o algoritmo ORB existe a apenas seis anos (o algoritmo foi desenvolvido por Rublee em 2011), não há muito material acadêmico comparativo sobre o mesmo.

Por estes motivos, o algoritmo ORB foi o algoritmo selecionado neste trabalho para obter e descrever os pontos chaves.

Capítulo 3

DESENVOLVIMENTO

Para atingir os objetivos descritos no capítulo 1 foram realizadas quatro etapas, sendo necessário o desenvolvimento de quatro aplicações. Todas as aplicações criadas para este trabalho foram desenvolvidas utilizando a linguagem Python 2.7 com auxílio da biblioteca OpenCV 3.0.0.

A primeira aplicação realiza o processo de detecção e descrição de pontos chaves em uma imagem. Uma segunda aplicação foi criada para processar um vídeo, realizando a leitura de cada *frame*, utilizando a primeira aplicação como módulo para detectar e descrever os pontos chaves dos *frames*. Além disso, essa aplicação executa os métodos de correspondência entre a imagem base e o *frame*, salvando em um arquivo no formato *txt* (texto) o tempo de execução da correspondência para cada *frame*. Esse algoritmo é executado vinte vezes, gerando múltiplas amostras de tempo de processamento da correspondência. Esses arquivos serão processados em seguida.

Uma terceira aplicação foi desenvolvida para realizar a leitura dos múltiplos arquivos gerados pela segunda e gerar um único arquivo contendo as médias de tempo para cada combinação entre algoritmo e quantidade de pontos.

Por último, uma quarta aplicação lê os arquivos, fornecendo informações mais enfatizadas a respeito das amostras. Essas informações serão analisadas no capítulo 4.

Este capítulo detalha como foi realizado o processo de desenvolvimento e demonstra o funcionamento dessas aplicações desenvolvidas.

3.1 Detecção de Pontos Chave e Extração dos Vetores Descritores

O primeiro passo do desenvolvimento deste trabalho consiste na detecção dos pontos chaves em uma imagem I para extrair os seus descritores I_d.

Para isso, um módulo foi implementado, recebendo como parâmetros uma imagem I e n, onde n é o número de pontos que serão encontrados e descritos pelo algoritmo ORB. A figura 3.1 ilustra o funcionamento deste módulo.

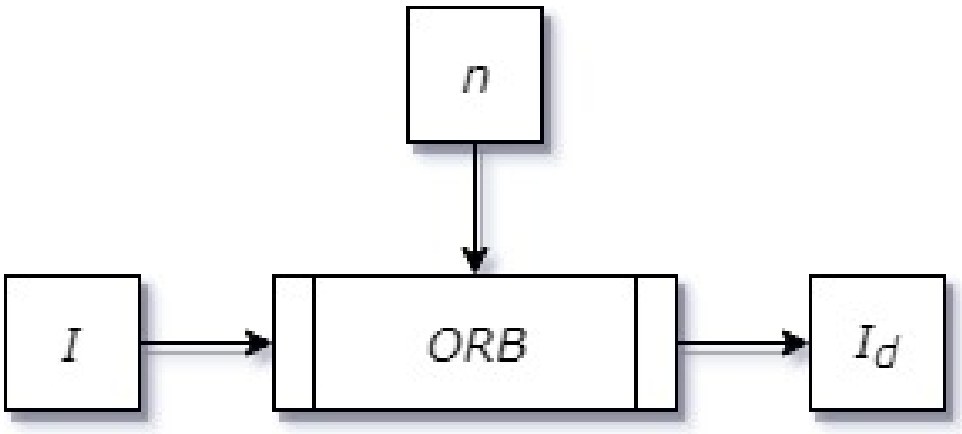

Figura 3.1 – Fluxograma da aplicação que detecta e descreve n pontos chave em uma imagem I e retorna o conjunto de descritores I_d.

Este módulo será executado em uma imagem base, assim como em cada *frame* de um vídeo de teste. O resultado deste módulo será um conjunto de pontos chave com seus respectivos descritores.

3.2 Correspondência de Pontos Chaves

Os pontos chave detectados e descritos através do algoritmo ORB anteriormente citado, serão utilizados para realizar a correspondência utilizando os algoritmos abordados no capítulo 2: *BFM*, *KNN* e *FLANN*.

Para isso, foi desenvolvido uma aplicação que utiliza o módulo descrito em 3.1 conforme ilustra o fluxograma na figura 3.2.

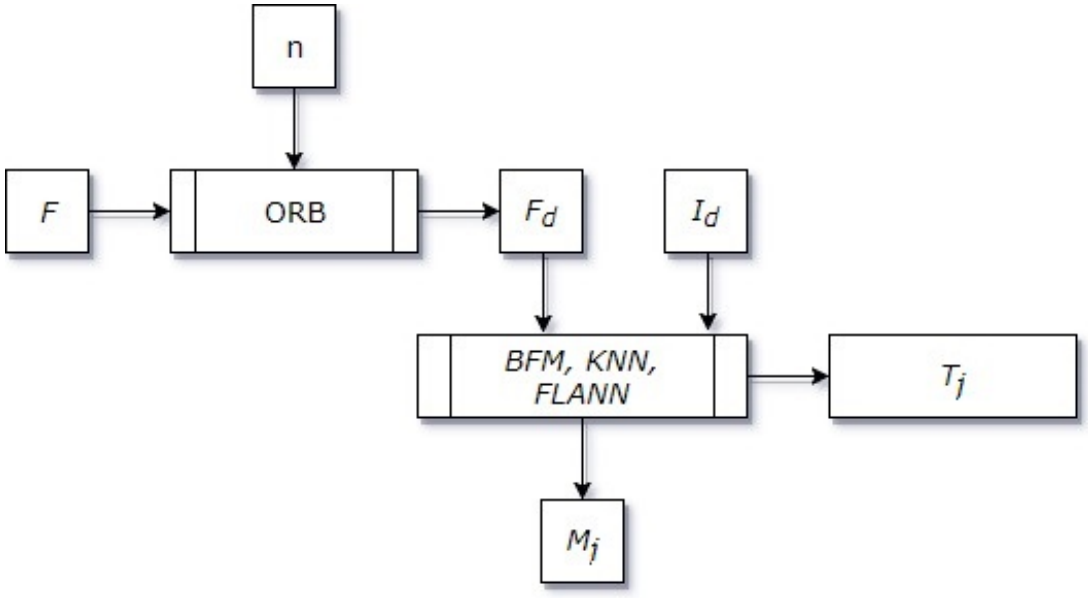

Figura 3.2 – Fluxograma da aplicação que realiza a correspondência pontos chaves.

Onde F representa um *frame* de um vídeo e F_d representa os vetores descritores daquele frame. Assim, os descritores F_d e I_d são utilizados para realizar a correspondência através de um dos três algoritmos, tendo como resultado T_j e M_j, sendo T_j o tempo percorrido durante a execução da correspondência e M_j as correspondências dos pontos chave.

Uma combinação A de algoritmo de correspondência (*BFM, KNN e FLANN*) e n gera um arquivo de amostras MT_i, de maneira que $A = \{MT_1, MT_2, MT_3, ..., MT_i\}$.

Para realizar a medição, a linguagem Python 2.7 fornece alguns módulos de medição de tempo, sendo estas *time.time*, *time.clock*, e *timeit*. A documentação do python define que na maioria dos casos, utilizar a medição através de *time.clock* será mais precisa que *time.time*, uma vez que a precisão *time.time* se dá pelo sistema operacional, ficando normalmente na casa dos segundos, enquanto a precisão de *time.clock* se dá em microsegundos.

Pelos motivos citados acima, o módulo *time.clock* foi o escolhido para a medição neste processo.

Ao final desse processo será gerado um conjunto $MT_i = \{T_1, T_2, T_3, ..., T_j\}$ contendo j medidas de tempo, considerando os fatores n, e o algoritmo de correspondência utilizado, sendo j a quantidade de *frames* existentes no vídeo.

Esse processo será repetido i vezes gerando um conjunto de amostras $A = \{MT_1, MT_2, ..., MT_i\}$ para cada combinação de n e algoritmo de correspondência.

3.3 Cálculo do Tempo Médio

Em seguida, para garantir a consistência das informações, o tempo médio das amostras A é calculado. Para isso, em cada grupo de amostras A, tira-se a média de cada $T_j \in MT_i$ sendo

$$A_m(T_j) = \frac{\sum_{k=1}^{i} MT_k(T_j)}{i}$$

onde A_m é o tempo médio das amostras $MT_i \in A$, de maneira que $A_m(T_j)$ é o valor médio de $MT_i(T_j)$. Após o final desse processo, um novo arquivo A_m será gerado com todos os valores de tempo médio de cada *frame*.

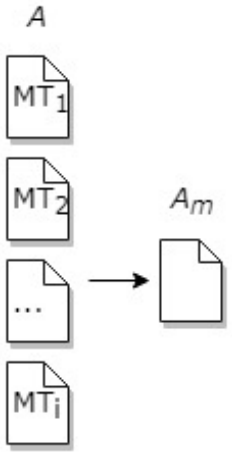

Figura 3.3 – Fluxograma da aplicação que realiza a correspondência pontos chaves.

Assim, a figura 3.3 demonstra o fluxo da aplicação responsável por ler os arquivos do grupo A e gravar um novo arquivo A_m contendo os valores de média.

Capítulo 4

TESTES E RESULTADOS

Este capítulo relata os resultados obtidos através dos testes realizados nas amostras obtidas através das aplicações desenvolvidas no capítulo 3 detalhando a maneira como as análises dos dados foram realizadas.

O primeiro passo foi realizar a obtenção das amostras, e em seguida, dois comparativos foram realizados: um comparativo simplificado através da média e um comparativo através da técnica de estatística ANOVA.

4.1 Obtenção das Amostras

A figura 4.1 foi a imagem selecionada para I, utilizada para extrair I_d, conforme abordado em 3.1.

Figura 4.1 – Imagem base *I* utilizada nas medições.

Conforme demonstrado em 3.1, uma aplicação foi desenvolvida, sendo exe-

cutadas as operações de correspondência de pontos chaves de maneira a extrair 30 amostras A, sendo 6 para cada algoritmo, onde $n \in (5, 10, 25, 50, 100, 200)$ e $i = 20$. Os algoritmos analisados foram *BFM* com as distâncias de *Hamming* e *Euclidiana*, o algoritmo *KNN* com as distâncias *Hamming* e *Euclidiana* e a biblioteca *FLANN* que utiliza a distância de *Hamming*. Para os algoritmos *KNN* e para a biblioteca *FLANN*, a quantidade k de vizinhos retornados selecionada foi de $k = 3$.

Em seguida, para cada amostra A foi calculada a sua média A_m do conjunto, de maneira que $A_m = \frac{\sum_{i=1}^{i=20} MTi}{20} = \{T_{m0}, T_{m1}, ..., T_{mj}\}$.

4.2 Comparação por Média

Na área de estatística, a média é utilizada para encontrar a concentração dos dados de uma distribuição como o ponto de equilibro das frequências de um histograma (PAIVA, 2016), sendo um valor significativo de uma lista de números.

A tabela 4.1, demonstra o tempo médio de processamento (em μs — microsegundos) de correspondência para o algoritmo *BFM*, utilizando as distâncias de *Hamming* e *Euclidiana*, aplicadas à 5, 10, 25, 50, 100 e 200 pontos chaves detectados com o algoritmo *ORB*.

Algoritmo BFM		
Quantidade de Pontos	**Hamming**	**Euclidiana**
5	$28,670\mu s$	$7,600\mu s$
10	$28,801\mu s$	$7,733\mu s$
25	$28,581\mu s$	$7,648\mu s$
50	$28,275\mu s$	$7,783\mu s$
100	$28,826\mu s$	$7,597\mu s$
200	$29,360\mu s$	$7,249\mu s$

Tabela 4.1 – Valores das médias para o algoritmo *BFM*, utilizando as distâncias Hamming e Euclidiana, em μs (microsegundos).

A tabela 4.2, demonstra o tempo médio de processamento de correspondência para o algoritmo *KNN*, utilizando as distâncias de *Hamming* e *Euclidiana*, aplicadas à 5, 10, 25, 50, 100 e 200 pontos chaves detectados com o algoritmo *ORB*.

A tabela 4.3, demonstra o tempo médio de processamento de correspondência para a biblioteca de algoritmos *FLANN*, que utiliza a distância de *Hamming*, aplicadas à 5, 10, 25, 50, 100 e 200 pontos chaves detectados com o algoritmo *ORB*.

Em seguida, comparou-se os algoritmos de três maneiras: Algoritmos agrupados por distância de *Hamming*; Algoritmos agrupados por distância *Euclidiana* e Algoritmos de maneira geral.

Algoritmo KNN		
Quantidade de Pontos	**Hamming**	**Euclidiana**
5	$10,254\mu s$	$4,586\mu s$
10	$8,766\mu s$	$4,467\mu s$
25	$8,700\mu s$	$4,469\mu s$
50	$8,695\mu s$	$4,452\mu s$
100	$8,663\mu s$	$4,418\mu s$
200	$8,581\mu s$	$4,499\mu s$

Tabela 4.2 – Valores das médias para o algoritmo *KNN*, utilizando as distâncias Hamming e Euclidiana, em μs (microsegundos).

Algoritmo FLANN	
Quantidade de Pontos	**Hamming**
5	$9,800\mu s$
10	$9,757\mu s$
25	$9,741\mu s$
50	$9,744\mu s$
100	$9,706\mu s$
200	$9,918\mu s$

Tabela 4.3 – Valores das médias para o algoritmo *FLANN*, que utiliza a distância de Hamming, em μs (microsegundos).

A figura 4.2 demonstra a média de tempo, agrupando os algoritmos que utilizaram a distância de *Hamming* em seu processamento, a figura 4.3 demonstra a média dos algoritmos que utilizaram a distancia *Euclidiana* e a figura 4.4 compara todos os algoritmos analisados de maneira geral.

Figura 4.2 – Média de tempo dos algoritmos que utilizam distância de Hamming no processo, em μs (microsegundos).

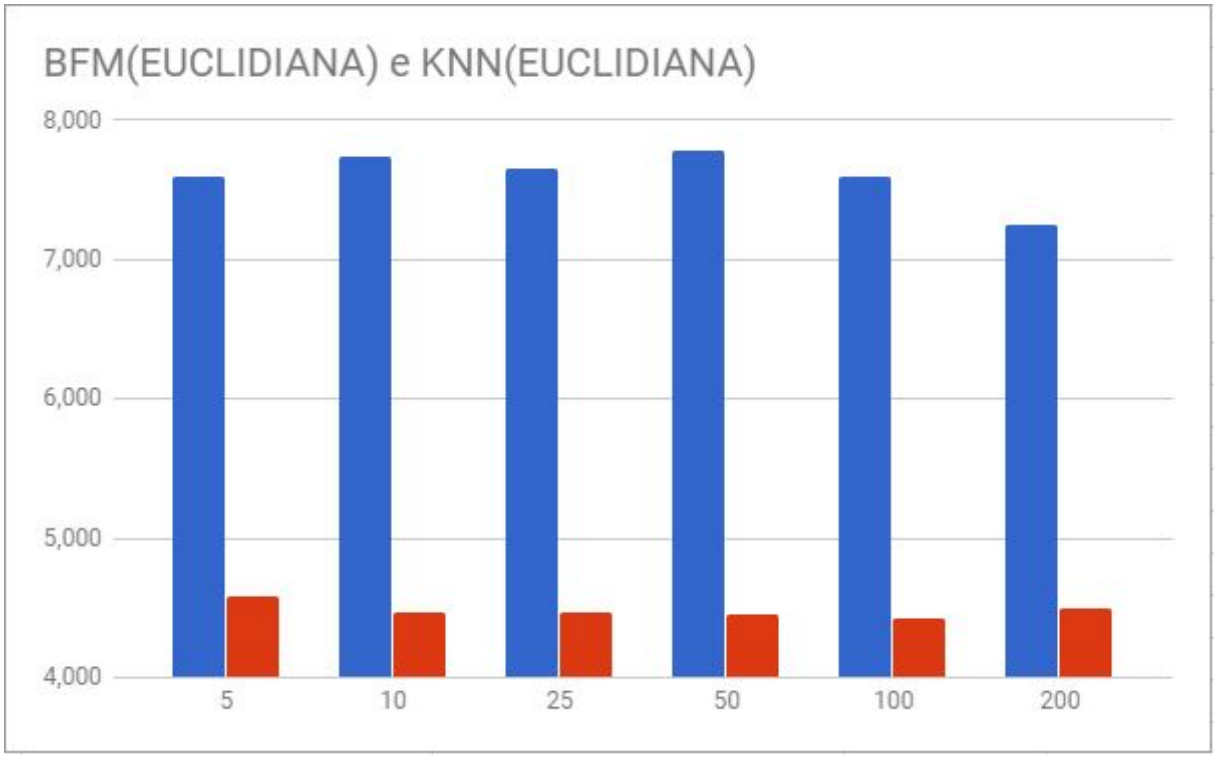

Figura 4.3 – Média de tempo dos algoritmos que utilizam distância Euclidiana no processo, em μs (microsegundos).

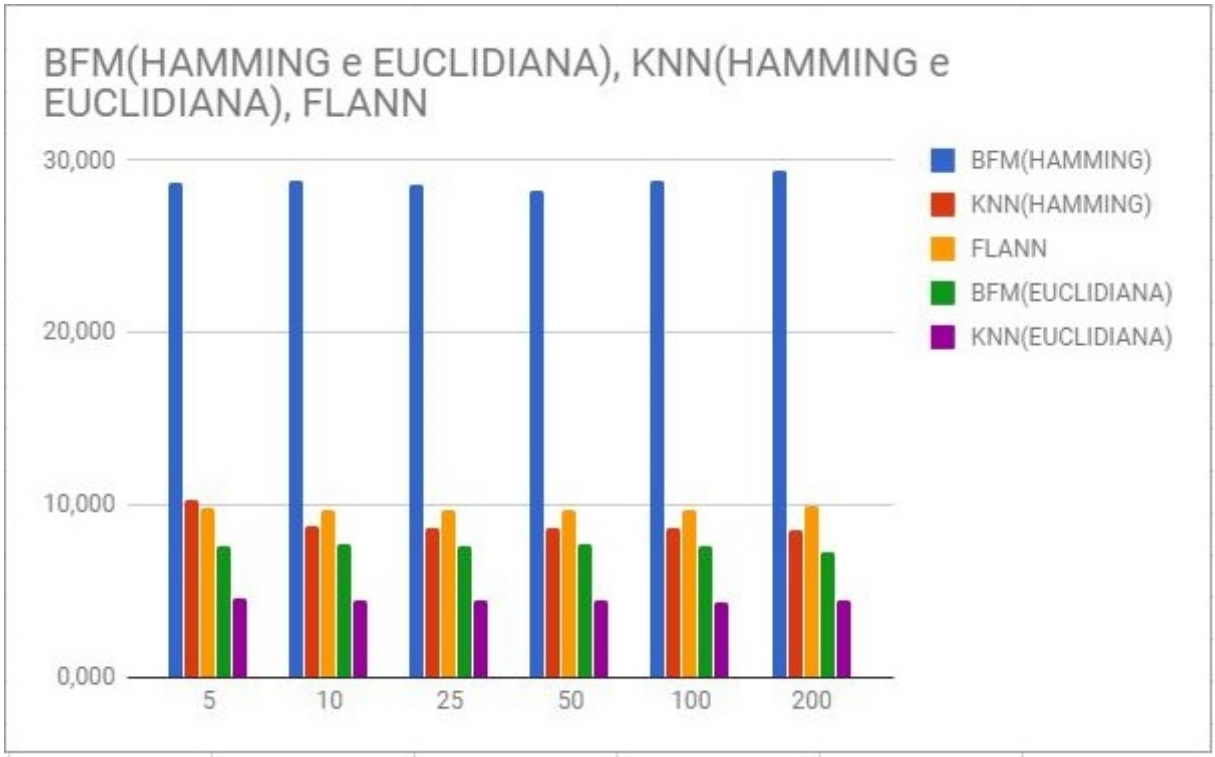

Figura 4.4 – Média de tempo geral dos algoritmos, em μs (microsegundos).

4.3 Comparação por ANOVA

ANOVA é uma técnica estatística de análise de variância de dados que permite avaliar as variações dos dados de conjuntos (MILONE, 2009).

Para realizar os cálculos, os dados são dispostos em uma matriz *m X n*, onde m é a quantidade de conjuntos e n é a quantidade de elementos de cada conjunto. Neste trabalho, $m = 5$ (*BFM* com distância de *Hamming*, *BFM* com distância *Euclidiana*, *KNN* com distância de *Hamming*, *KNN* com distância *Euclidiana* e *FLANN*) e $n = 373$, sendo 373 a quantidade de *frames* existentes no vídeo utilizado.

O primeiro passo é calcular o valor da média das médias, denotada por $\bar{x}$. A média das médias é definida por

$$\bar{x} = \frac{\sum_{i,j=1}^{m,n} E(i,j)}{mn}$$

onde $E(i,j)$ é o valor do elemento na matriz *m X n* nas posições i e j. Os valores encontrados são demonstrados na tabela 4.4.

Um vez calculado $\bar{x}$, é possível calcular o numerador *SQT* (Soma dos Quadrados Total), definido por

Média das Médias	
Quantidade de Pontos	$\bar{x}$
5	$12,182\mu s$
10	$11,905\mu s$
25	$11,828\mu s$
50	$11,790\mu s$
100	$11,842\mu s$
200	$11,921\mu s$

Tabela 4.4 – Valores das médias das médias, em μs (microsegundos).

$$SQT = \sum_{i,j=1}^{m,n} (E(i,j) - \bar{x})^2$$

assim como os respectivos graus de liberdade *GL* para *SQT*, definida por $GL_{SQT} = m.n - 1 = 1864$. Assim, para cada agrupamento de conjuntos por quantidade de pontos, encontra-se os valores de *SQT* demonstrados na tabela 4.5.

Soma dos Quadrados Total		
Quantidade de Pontos	*SQT*	GL_{SQT}
5	$0,410\mu s$	1864
10	$0,154\mu s$	1864
25	$0,137\mu s$	1864
50	$0,133\mu s$	1864
100	$0,151\mu s$	1864
200	$0,156\mu s$	1864

Tabela 4.5 – Soma dos Quadrados Total e seus respectivos Graus de Liberdade.

Pode existir variâncias para todo o conjunto de $m.n$ elementos, entretanto, tal variância pode ser advinda de elementos de um mesmo grupo ou advindas de grupos diferentes (MILONE, 2009). Para isso, é necessário dividir *SQT* entre *SQD* (Soma dos Quadrados Dentro) e *SQE* (Soma dos Quadrados Entre).

Enquanto que *SQT* é a distância entre os dados e $\bar{x}$, *SQD* é a distância dos entre os dados de um grupo em relação à média daquele respectivo grupo (sendo um grupo, uma coluna na matriz *m X n*) e *SQE* é a distância entre a média de um grupo em relação à média das médias. Assim, *SQD* é denotada por

$$SQD = \sum_{i,j=1}^{m,n} (E(i,j) - M(i))^2$$

onde $M(i)$ é a média do grupo do elemento $E(i,j)$. Calcula-se então $GL_{SQD} = m.(n-1) = 1860$, conforme os valores demonstrados na tabela 4.6.

Soma dos Quadrados Dentro		
Quantidade de Pontos	*SQD*	GL_{SQD}
5	$0,276\mu s$	1860
10	$0,015\mu s$	1860
25	$0,000\mu s$	1860
50	$0,001\mu s$	1860
100	$0,011\mu s$	1860
200	$0,008\mu s$	1860

Tabela 4.6 – Soma dos Quadrados Dentro dos grupos e seus respectivos Graus de Liberdade.

Enquanto *SQT* e *SQD* realizam o cálculo baseado diretamente nos valores da matriz, *SQE* calcula para cada elemento da matriz a distância da média do grupo daquele elemento em relação à média das médias, de forma que

$$SQD = \sum_{i,j=1}^{m,n} (M(i) - \bar{x})^2 = \sum_{i=1}^{m} n.(M(i) - \bar{x})^2$$

com os respectivos graus de liberdade $GL_{SQE} = m - 1 = 4$. Este passo resulta na tabela 4.7.

Soma dos Quadrados Entre		
Quantidade de Pontos	*SQE*	GL_{SQE}
5	$0,134\mu s$	4
10	$0,139\mu s$	4
25	$0,136\mu s$	4
50	$0,132\mu s$	4
100	$0,140\mu s$	4
200	$0,147\mu s$	4

Tabela 4.7 – Soma dos Quadrados Entre os grupos e seus respectivos Graus de Liberdade.

Nota-se então que $SQT = SQD + SQE$, assim como $GL_{SQT} = GL_{SQD} + GL_{SQE}$, conforme discutido por Milone (2009). Para realizar a análise estatística, calcula-se então a variância das somas, definidas por

$$V_{SQT} = \frac{SQT}{GL_{SQT}}$$

$$V_{SQD} = \frac{SQD}{GL_{SQD}}$$

$$V_{SQE} = \frac{SQE}{GL_{SQE}}$$

definindo assim a tabela 4.8 de variâncias para os grupos, onde é possível notar que os valores de V_{SQT} e $V)SQD$ são muito baixos em todas as medições, em alguns casos, se aproximando de zero.

Variâncias			
Quantidade de Pontos	V_{SQT}	V_{SQD}	V_{SQE}
5	0,0002202	0,0001485	0,0335633
10	0,0000829	0,0000084	0,0347537
25	0,0000738	0,0000005	0,0341708
50	0,0000717	0,0000006	0,0331422
100	0,0000815	0,0000062	0,0350875
200	0,0000839	0,0000047	0,0369472

Tabela 4.8 – Valores de Variâncias.

Para realizar a análise foram utilizadas duas hipóteses. A primeira é a hipótese denotada por H_0 cuja premissa é a de que a escolha do algoritmo não faz diferença. A outra hipótese, representada por H_1, define que a escolha do algoritmo faz diferença.

Para testar estas hipóteses, foi selecionado o teste de hipótese F proposto por Fisher (1922) que define que

$$F = \frac{V_{SQE}}{V_{SQD}}$$

Assim, torna-se possível montar a tabela 4.9, que demonstra os valores obtidos para F para cada análise.

Estatística F	
Quantidade de Pontos	F
5	226,077
10	4161,260
25	74698,743
50	55925,399
100	5670,279
200	7945,039

Tabela 4.9 – Soma dos Quadrados Entre os grupos e seus respectivos Graus de Liberdade.

Ao realizar um teste de hipótese, é necessário selecionar um nível de significância para analisar as hipóteses, denotado por α. Neste trabalho, $\alpha = 0,01$ (1%), o que significa que ao considerar a hipótese H_0 como verdadeira, se houver uma chance menor que 1% de se obter essa estatística F, a hipótese H_0 será rejeitada (FISHER, 1922).

Para isso, utiliza-se um valor crítico denotado por F_c. Tanto o numerador quanto o denominador deste trabalho se encontram abaixo de zero, por isso, de acordo com a

tabela publicada por Dinov (2002) para $\alpha = 0,01$, utilizaremos os valores 1 em ambos, tendo como valor crítico $F_c = 4052.181$.

Assim, onde $F > F_c$, a hipótese H_0 é rejeitada, uma vez que a probabilidade de se obter o valor F de maneira aleatória na matriz é muito pequena, o que significa que a hipótese H_1 que define que os algoritmos selecionados afetam o desempenho da correspondência é a hipótese correta. Dessa maneira, pode-se afirmar que quando a quantidade de pontos for maior ou igual à dez ($n \geq 10$), o algoritmo utilizado influencia no tempo de execução.

Capítulo 5
CONCLUSÃO

Na visão computacional, o desempenho dos algoritmos utilizados afeta diretamente o resultado de aplicações de tempo real. Isso ocorre principalmente em áreas como *VANT* (Veículos Aéreos Não Tripulados — normalmente drones), onde o processamento é feito de forma contínua exigindo dos algoritmos um tempo de resposta mais rápido possível, uma vez que atrasos no processamento podem causar problemas como colisão e, consequentemente, queda.

Uma das funções utilizadas neste contexto da visão computacional é a correspondência de pontos chave, responsável por identificar pontos congruentes entre duas imagens distintas. Para realizar a correspondência, é necessário antes identificar e descrever os pontos chaves. Vários métodos e técnicas foram desenvolvidos para resolver o problema de detecção e descrição de pontos chave, e algoritmos como *SIFT*, *SURF* e *ORB* ganharam notoriedade devido à sua confiabilidade e precisão. Dentre as aplicações em tempo real, o algoritmo *ORB* se destacou por seu alto desempenho, sendo então amplamente utilizado.

Os algoritmos de correspondência todavia são comumente escolhidos de maneira empírica, possuindo pouco, ou nenhum embasamento teórico.

Foi apresentada neste trabalho a implementação de três dos principais algoritmos de correspondência, aplicados em pontos chaves extraídos através do algoritmo *ORB* com o objetivo de medir o tempo de execução de cada um e fornecer embasamento para a decisão sobre qual método de correspondência utilizar em aplicações de tempo real.

Os resultados das medições indicam que a quantidade de pontos não possui tanta influência no desempenho de aplicações em tempo real quanto o algoritmo selecionado.

Conforme testes realizados no capítulo 3.3, é possível afirmar que a utilização do algoritmo de correspondência *BFM* (*Brute Force Matching*), utilizando a distância

de *Hamming* para a medição, não é recomendada para os pontos chave detectados e descritos pelo algoritmo ORB. Uma vez que o algoritmo *BFM* compara um vetor descritor de uma imagem I_a com todos os outros vetores descritores de uma imagem I_b, e a distância de *Hamming* verifica a distância modular de todos os itens do vetor da imagem I_a com cada vetor comparado em I_b, a complexidade dessa combinação influência diretamente no desempenho da execução, independente da quantidade de pontos comparados.

Em 4.3, utilizando a comparação estatística ANOVA, de acordo com a tabela 4.9, quando a quantidade de pontos chave desejada for maior ou igual a dez ($n \geq 10$), haverá influência no desempenho da aplicação por parte do algoritmo de correspondência.

Assim, é possível afirmar que, dentre os algoritmos analisados, quando a quantidade de pontos chave desejada for menos de 10 ($n < 10$), qualquer algoritmo pode ser utilizado, já quando a quantidade de pontos chave for maior ou igual à dez ($n \geq 10$), é necessário cautela quanto ao algoritmo utilizado. Assim, conforme é possível observar nos gráficos do capítulo 3.3, o algoritmo *KNN* com distância Euclidiana mostrou melhor desempenho, ao realizar a busca considerando *k = 3* (quantidade de vizinhos retornados).

Dessa maneira, considerando uma aplicação em tempo real que captura 24 quadros por segundo, após um período de 24 horas (ou 86.400 segundos), terá realizado a leitura de 2.073.600 quadros. Com base nas tabelas 4.1 e 4.2, essa aplicação, realizando a comparação de 10 pontos chave utilizando o algoritmo *BFM* com distância de *Hamming*, após dado período de 24 horas, terá um atraso de 59,72 segundos. Essa mesma aplicação, ao substituir o algoritmo de correspondência para o algoritmo *KNN* com distância Euclidiana, terá um atraso de 9,26 segundos. Neste cenário, a substituição do algoritmo *BFM* com distância de *Hamming* pelo algoritmo *KNN* com distância Euclidiana seria 6,4 vezes mais rápido a mesma tarefa.

Assim, com o desenvolvimento deste trabalho é possível perceber que melhorias no desempenho de aplicações tem tempo real podem ser realizadas através de uma escolha minuciosa dos algoritmos de correspondência utilizados.

Neste trabalho, o desempenho de três algoritmos específicos de correspondência de pontos chave foi analisada, possibilitando então em trabalhos futuros comparar o desempenho de novos algoritmos de correspondência que venham a surgir.

Uma vez que o presente trabalho analisou o tempo de desempenho dos algoritmos, outra proposta para possíveis trabalhos futuros seria realizar a comparação da qualidade da detecção dos pontos chaves. Desta forma, será possível gerar um índice de custo/benefício dos algoritmos estudados.

REFERÊNCIAS

ARYA, S. et al. “an optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *ACM, vol 45*, 1998.

BAWA, M.; CONDIES, T.; GANESAN, T. Lsh forest: Selft tunning indexes for similarity search. *Proc. 14th Int. Conf. World Wide Web*, 2005.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. SURF: Speeded up robust features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 3951 LNCS, p. 404–417, 2006. ISSN 03029743.

BEIS, J. S.; LOWE, D. G. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. *Proc. IEEE Conf. Coput. Vis. Pattern Recog.*, 1997.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res., vol. 13*, 2012.

CALONDER, M. et al. BRIEF: Binary robust independent elementary features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 6314 LNCS, n. PART 4, p. 778–792, 2010. ISSN 03029743.

CANNY, J. *A Computational Approach to Edge Detection*. [S.l.]: IEEE, 1986. 20 p.

DINOV, I. D. *F Distribution Tables*: Statistics online computational resource (socr). 2002. Disponível em: <http://www.socr.ucla.edu/applets.dir/f_table.html#FTable0.001>.

FISHER, R. A. On the interpretation of x^2 from contingency tables, and the calculation of p. *Journal of the Royal Statistical Society*, 1922.

GONZALES, R. C.; WOODS, R. E. *Processamento Digital de Imagens*: 3ª edição. [S.l.: s.n.], 2011.

HAJEBI, K. et al. Fast approximate nearest-neighbor search with k-nearest neighbor graph. *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011.

HARRIS, C.; STEPHENS, M. A combined corner and edge detector. *Alvey Vision Conference*, 1988.

HARTLEY, R.; SILPA-ANAN, C. “optimised kd-trees for fast image descriptor matching. *Proc. IEEE Conf. Comput. Vis. Pattern Recog*, 2008.

HOWARD, A. *Álgebra Linear com Aplicações*: 8ª edição. [S.l.: s.n.], 2001. ISBN 978-85-7307-847-3.

KIRSCH, R. Computer determination of the constituent structure of biological images. *Computers and Biomedical Research*, 1971.

LINDEBERG, T. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 1998.

LINDEBERG, T. Scale-space. *Encyclopedia of Computer Science and Engineering*, 2008.

LINDEBERG, T. Scale selection properties of generalized scale-space interest point detectors. *Journal of Mathematical Imaging and Vision, Volume 46*, 2013. Disponível em: <https://link.springer.com/article/10.1007%2Fs10851-012-0378-3>.

LOWE, D. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, p. 1150–1157 vol.2, 1999. ISSN 0-7695-0164-8. Disponível em: <http://ieeexplore.ieee.org/document/790410/>.

LOWE, D. G. Distinctive Image Features from. *International Journal of Computer Vision*, v. 60, n. 2, p. 91–110, 2004. ISSN 0920-5691.

MILONE, G. *Estatística geral e aplicada*. [S.l.: s.n.], 2009. ISBN 85-221-0339-9.

MUJA, M.; LOWE, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In: *International Conference on Computer Vision Theory and Application VISSAPP'09)*. [S.l.]: INSTICC Press, 2009. p. 331–340.

MUJA, M.; LOWE, D. G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 36*, 2014.

NELDER, J. A.; MEAD, R. A simplex method for function minimization. *Comput. J., Vol. 7*, 1965.

NOVAIS, J. P. Aplicação dos algoritmos sift e surf na classificação de sub-imagens por discriminação de textura. 2016.

PAIVA, M. Matemática e suas tecnologias: Fb vestibular. 2016.

PATEL, A. et al. Performance analysis of various feature detector and descriptor for real-time video based face tracking. *International Journal of Computer Applications*, 2014.

ROSIN, P. L. Measuring corner properties. *Computer Vision and Image Understanding*, 1999.

ROSTEN, E.; DRUMMOND, T. Machine learning for high-speed corner detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 3951 LNCS, p. 430–443, 2006. ISSN 16113349.

RUBLEE, E. et al. ORB: An efficient alternative to SIFT or SURF. *Proceedings of the IEEE International Conference on Computer Vision*, p. 2564–2571, 2011. ISSN 1550-5499.

SEBASTIAN, T. B.; KIMIA, B. B. Metric-based shape retrieval in large databases. *Proc. IEEE Conf. Comput. Vis. Pattern Recog*, 2002.

SHAPIRO, L. G.; STOCKMAN, G. C. Computer vision. *Prentice Books, Upper Saddle River*, 2001.

SMITH, S. M.; BRADY, J. M. SUSAN - A New Approach to Low Level Image Processing. v. 1, p. 59, 1995.

SWETHA, R.; VEENA, N.; ATHAVALE, P. A. A Comparison of Canny , Robert edge detection filters and their superposition NC3PS-2016. p. 213–218, 2016.

TANEMBAUM, A. S. *Organização Estruturada de Computadores*. [S.l.: s.n.], 1990. ISBN 85-216-1170-6.

VALE, G. D.; POZ, A. D. Processo de detecção de bordas de Canny. *Boletim de Ciências Geodésicas*, p. 67–78, 2004. Disponível em: <http://ojs.c3sl.ufpr.br/ojs-2.2.4/index.php/bcg/article/viewArticle/1421>.

WILLIS, A.; SUI, Y. An algebraic model for fast corner detection. *IEEE 12th International Conference on Computer Vision*, 2009.

Apêndice A

CÓDIGOS E IMAGENS

Os códigos das aplicações desenvolvidas para atingir os objetivos deste trabalho se encontram disponíveis em http://github.com/brudev/research.

Além disso, em https://github.com/brudev/research também se encontram disponíveis as imagens e vídeos utilizados, assim como os resultados dos experimentos.