Implementação da *Image Matching*: uma nova solução para um problema NP-completo

1st Levy Santiago
Universidade Federal da Bahia (UFBA)
Salvador, Brasil
levy.santiago@ufba.br

2nd Marcos Pinheiro
Universidade Federal da Bahia (UFBA)
Salvador, Brasil
marcos.pinheiro@ufba.br

3rd Rita Barretto
Universidade Federal da Bahia (UFBA)
Salvador, Brasil
rita.barretto@ufba.br

***Resumo*—A questão P $\neq$ NPC é intrigante e complexa para a teoria da ciência da computação. Diante desse cenário, este trabalho abordou o problema NP-completo *Image Matching* (IM) e trouxe uma nova forma de implementação para ele. A metodologia adotada partiu do mapeamento dos *pixels* das imagens e subsequente associação das semelhanças entre eles, considerando o deslocamento de *pixels*. Na sequência, foi realizada uma comparação dos resultados obtidos pelos novos algoritmos, a saber, lr2m e lr2mRGB, com os obtidos pelo *Scale Invariant Feature Transform* (SIFT), destacando-se o tempo de execução, bem como a qualidade e a fidedignidade das semelhanças identificadas. Observou-se uma grande variação nos tempos de resposta de lr2m e lr2mRGB. Embora o lr2m utilize *pixels* em *grayvalues* e o lr2mRGB, *pixels* em RGB, ambos apresentaram tempos de resposta muito próximos para os testes: T1 e T3. Nos outros casos de testes, porém, observou-se uma diferença a favor do lr2m com relação ao lr2RGB. Em todos os testes, constatou-se o melhor desempenho do SIFT se comparado ao lr2m e lr2mRGB.**

***Index Terms*—algoritmo, correspondência de imagem, *image matching*, np-completo, implementação**

I. Introdução

A questão P $\neq$ NPC (NP-completo) é extremamente intrigante e complexa para a teoria da ciência da computação. P é uma classe de problemas que podem ser resolvidos em tempo polinomial e NPC, uma classe de problemas considerados intratáveis pela maioria dos teóricos [1]. Portanto, ainda não foi descoberto nenhum algoritmo de tempo polinomial para um problema NP-completo, bem como não foi provado que não pode existir nenhum algoritmo de tempo polinomial para eles. Diante desse cenário, este trabalho aborda o problema da *Image Matching* (IM) ou Correspondência de Imagem, inerente ao reconhecimento de imagem e tido como um problema difícil sob a perspectiva do algoritmo, ou seja, NP-completo [2].

A IM visa à identificação da correspondência de estrutura/-conteúdo igual ou semelhante de duas ou mais imagens [3]. Assim, pretende-se trazer uma nova forma de implementação e resolução para o problema da IM, de maneira que a mencionada implementação não necessariamente seja melhor ou mais rápida do que as propostas existentes atualmente, mas que demonstre um novo modo de pensar na resolução para o problema.

Na literatura, encontram-se diversos métodos e técnicas de comparação de imagens tradicionais e amplamente adotados, destacando-se: método baseado em área [3] e método baseado em recursos [4]. Adicionalmente, existem o algoritmo *Scale Invariant Feature Transform* (SIFT) [5], a comparação de histogramas [6] e a redução da imagem a um código *hash* [7]. Atualmente, a aplicação de técnicas de *deep learning* para representação de informações de imagem e/ou medição de similaridade, bem como regressão de parâmetros de transformação de pares de imagens, são muito importantes para a comunidade de processamento de imagens e visão computacional.

A nova implementação proposta neste trabalho, por sua vez, adota a metodologia que parte do mapeamento dos *pixels* das imagens e associa as semelhanças entre elas. É realizada uma avaliação, dos resultados obtidos pelo novo algoritmo comparados com os resultados obtidos por meio do algoritmo SIFT [5], verificando-se o tempo de execução, da análise das imagens, bem como a qualidade e a fidedignidade das semelhanças identificadas a partir de cinco casos de teste.

O conteúdo do artigo está organizado em seis seções. A seção I é a introdução, que destaca o escopo e a estrutura do artigo. A seção II apresenta uma definição do problema da IM, sua formalização e complexidade. A seção III é o referencial teórico apresentando trabalhos relacionados. A seção IV apresenta a metodologia de implementação e como foi realizada sua avaliação. A seção VI demonstra a complexidade assintótica, análise de desempenho e qualidade dos resultados gerados pelos algoritmos, a aplicação e comparação dos algoritmos. Por fim, a seção VII resume as discussões e aponta desdobramentos futuros.

II. *Image Matching*

O problema da IM tem sido abordado por diversos autores [3], [8]–[10], a fim de buscar melhores técnicas para uma solução mais rápida e precisa. O problema consiste em analisar duas imagens **A** e **B**, onde **B** é uma transformação geométrica de **A**. O resultado desta análise deve ser o encontro de características semelhantes entre elas, de forma que, no final, estas correspondências sejam realçadas e apresentadas para facilitar uma determinada investigação [11]. A IM é uma componente-chave para diversos processos de análise de imagens e é muito importante para inúmeras aplicações, como navegação, orientação, vigilância automática, fotogrametria e visão robótica [12]. Na medicina, por exemplo, é de grande

importância para encontrar relações entre diferenças no posicionamento do paciente, modalidades e aquisição de imagens variadas [13].

A. Formalização

Existe uma variedade de algoritmos para solucionar este problema, a exemplo de [14], que é citado por [2] para ajudar na formalização do IM. Basicamente, o problema deve receber como entrada (instância) duas imagens de tamanho M x M, onde a segunda imagem é uma transformação da primeira. A solução deve retornar um mapeamento destas imagens de forma que sejam destacadas as semelhanças entre elas, ou seja, deve existir uma função tal que: $f : MxM \rightarrow MxM$. As duas imagens possuem valores em cinza (*grayvalues*) que fazem parte de um alfabeto finito $\alpha = \{1, ..., G\}$. Além disso, são definidas duas funções de distância, d_α, que mede a correspondência em *grayvalues* e d_d, medindo a distorção introduzida pela correspondência. Assim, é possível formalizar um problema de decisão correspondente, que é definido pela seguinte pergunta: tendo uma instância de IM e um limite de custo c', existe uma função de mapeamento tal que $c(A, B, f) \leq c'$?", sendo que o custo é definido pela expressão ilustrada na Equação 1 [2].

$$
\begin{aligned}
c(A, B, f) = & \sum_{(i,j) \in M \times M} d_g(A_{i,j}, B_{f(i,j))}) \\
& + \sum_{(i,j) \in 1,...,M-1) \times M} d_d(f((i,j) + (1.0)) - f((i,j) + (1.0))) \\
& + \sum_{(i,j) \in M \times 1,...,M-1)} d_d(f((i,j) + (1.0)) - f((i,j) + (1.0)))
\end{aligned} \tag{1}
$$

B. Complexidade

O problema da IM é de otimização e NP-completo [2]. Ele é assim classificado porque pode ser verificado em tempo polinomial, a partir do problema de decisão $c(A, B, f)$ citado anteriormente, e pode ser construída uma redução do problema 3-SAT (problema de satisfatibilidade booleana) para o problema em questão. Esta redução é construída a partir da definição de uma instância para o 3-SAT. Uma vez definida a instância, é criado um grafo de dependência, o qual é então representado em um plano 2D (dimensão 2) e aprimorado para representar o comportamento lógico da instância, gerando duas imagens. Por fim, são definidos valores verdadeiros ('*true*') para os *pixels* que possuem uma correspondência e valores falsos ('*false*'), caso contrário, para satisfazer as cláusulas do problema.

III. Revisão de Literatura

Quando se trata de metodologias para comparação de imagens, várias são as técnicas utilizadas no problema. O algoritmo [5] detecta e cria descritores locais relativos a uma determinada imagem, sendo este descritor relativamente invariante a transformações de rotação, iluminação, escala e baixa variação de perspectiva. De uma maneira geral, este algoritmo é divido em quatro etapas principais [15]:

- Detecção de extremos;
- Localização de pontos-chave;
- Definição de Orientação;
- Descritor dos pontos-chave.

Considerando duas imagens i_1 e i_2 diferentes, que representam uma mesma cena, obtém-se com o SIFT duas matrizes M_1 e M_2, as quais contêm descritores de pontos-chave de i_1 e i_2, respectivamente. Para cada descritor de M_1, calcula-se a distância euclidiana com todos os descritores de M_2, comparando,assim, as imagens.

A comparação de histogramas é uma outra maneira de se obter informações das imagens [6]. Um histograma é definido como a quantificação do número de *pixels* para cada valor de intensidade de uma imagem ou uma representação gráfica da distribuição de intensidade [16]. A visão estatística sobre a distribuição dos *pixels* e dos níveis de brilho e contraste oferecida pelo histograma são variáveis úteis para comparação de imagens.

Outra metodologia seria garantir a integridade através do código *hash*. Isso leva à precisa identificação de uma imagem que teve sua sequência de bits alterada. Para tanto, cada imagem é reduzida a um pequeno código, identificando recursos destacados no arquivo de imagem original e misturando uma representação compacta desses recursos. Em [7], é apresentada uma técnica, utilizando *hashing*, onde, inicialmente, se processa a imagem reduzindo-a para o tamanho 256 × 256. Após isso, converte-se a imagem para escala de cinza, depois, extrai-se um vetor de características. A partir dele, constrói-se um *hashing* de múltiplas tabelas que é utilizado em comparações com outras imagens.

IV. Metodologia

A proposta é implementar um algoritmo na linguagem Python. Esta linguagem foi escolhida pelo fato de oferecer um bom desempenho, além de diversas ferramentas que ajudam nas implementações de algoritmos para reconhecimento de imagens [17]–[19]. Ela também oferece bibliotecas que resolvem o problema da IM, as quais, logicamente, não foram utilizadas na implementação do algoritmo deste projeto, uma vez que, o seu objetivo é criar uma nova implementação. Entretanto, foi escolhido um deles para realizar a comparação com o algoritmo proposto.

A primeira fase da estratégia é realizar um mapeamento dos *pixels* de uma das duas imagens. Este mapeamento gera uma lista que permite ter algumas características das imagens, a exemplo do código da cor e localização i, j de cada *pixel* da imagem. A posição dos *pixels* é utilizada para saber a exata localização no plano da imagem e, assim, possibilitar o cálculo das distâncias entre *pixels*. Identificando-se a cor de cada *pixel*, pode-se determinar possíveis objetos e respectivas cores que os caracterizam. Uma vez mapeada a primeira imagem, realiza-se o mesmo mapeamento na segunda, para, em seguida, associar possíveis semelhanças entre os mapas de cada uma. As semelhanças são tanto um conjunto de *pixels* que possuem o mesmo espectro de cores nas duas imagens, quanto a localização deslocada de *pixels* dos *pixels* na primeira imagem.

Primeiramente, a implementação utiliza os *grayvalues*, assim, obtém-se apenas um valor da cor da imagem (de 0 a 255). Mas, também traz a mesma implementação no caso de utilizar uma imagem colorida, sendo necessária a realização de alguns cálculos para considerar os três valores RGB (vermelho, verde e azul).

O pseudocódigo abaixo (Algoritmo 1) organiza os passos básicos que a implementação seguiu. O algoritmo recebe como entrada duas imagens **A** e **B** de mesmo tamanho MxM. O primeiro passo é obter os *pixels* das imagens **A** e **B** em escalas de cinza (`gvA` e `gvB`). Depois desta etapa, é realizado o mapeamento da imagem **A** (mapA), ou seja, o algoritmo percorre todos os *pixels* (i, j) em busca de valores menores do que um limite (que poderá também ser de 0 a 255). Este limite é definido para capturar *pixels* mais escuros da imagem, a fim de tentar encontrar objetos que estejam na imagem, desconsiderando então o *background*. Diferentemente do algoritmo apresentado no relatório parcial[1], foi aproveitado o mesmo método `selectPixels()` para já retornar o primeiro *pixel* do mapa de A (`ipA`), pois a partir deste, são estimadas as localizações dos outros *pixels* na imagem **B** de forma deslocada, e o *pixel* inicial de **B** (`ipB`), ou seja, o *pixel* que deve ser o primeiro *pixel* de **B** menor do que o `limit`. Com o `mapA`, `ipA` e `ipB`, pode-se calcular o mapa de **B** (`mapB`), o qual é obtido realizando alguns cálculos com o `mapA` a partir do `ipA`, a fim de estimar as possíveis posições dos mesmos *pixels* na imagem **B**, pois, como espera-se que **B** seja uma transformação da imagem **A**, deve conter os mesmos *pixels*, porém transladados para outra posição.

```
1 lr2Matching(A,B):
2   gvA <- grayvalues(A)
3   gvB <- grayvalues(B)
4   mapA, ipA, ipB <- selectPixels(gvA, limit)
5   mapB <- selectPixelsBy(mapA,ipA,ipB)
6   highlightPixels(A,B,mapA,mapB,gvA,gvB,limit)
```

Algoritmo 1: Solução proposta em pseudocódigo

A estratégia para mapear os *pixels* da segunda imagem (linha 5 do Algoritmo 1) considera a seguinte abordagem: percorrendo o `mapA` e, utilizando o `ipA` e `ipB`, é realizado um cálculo para predizer as possíveis posições dos mesmos *pixels* na imagem **B**. A partir da diferença de linha (i) e coluna (j) do *pixel* que está sendo percorrido (p) com o i, j de `ipA`, somada com o i, j de `ipB`, obtém-se a possível localização do mesmo *pixel* da imagem **A** em **B** (pxB_i, pxB_j). Assim, tem-se:

- $pxB_i = ipB_i + (p_i - ipA_i)$
- $pxB_j = ipB_j + (p_j - ipA_j)$

Então, a dupla i e j obtida é armazenada em um vetor, que será o `mapB`, juntamente com o valor da cor do *pixel*. Depois de obtido o `mapB`, são realizadas as marcações na imagem das correspondências encontradas, finalizando assim o algoritmo. Existe também a possibilidade de a imagem **B**, além de transladada, ter pequenas diferenças nos valores dos *pixels*. Por isso, é utilizado um valor limite na etapa de `highlightPixels()` para que os cálculos considerem os *pixels* em que a diferença dos seus valores devem ser menor do que este limite, ou seja: $|gvA(i, j) - gvB(i, j)| < limit$.

Adicionalmente, seguindo basicamente o mesmo fluxo, foi planejado o desenvolvimento do algoritmo utilizando os valores em RGB, a fim de ter mais um caso para, não só comparar com o SIFT, mas comparar com a própria implementação do código com *grayvalues* e verificar qual deles obteve um melhor desempenho e/ou qualidade no encontro de correspondências entre as imagens. Neste caso, como em RGB é preciso lidar com três valores para as três cores, na etapa de selecionar os *pixels* e obter o `mapA`, foi realizada a média destes três valores e aplicado um *limit* maior do que o algoritmo anterior, por ser um valor de comparação também maior. Neste caso, tem-se:

- $mediaA = (pxA(i, j)_R + pxA(i, j)_G + pxA(i, j)_B)/3$
- $mediaB = (pxB(i, j)_R + pxB(i, j)_G + pxB(i, j)_B)/3$
- $|mediaA - mediaB| < limit$

A avaliação da implementação foi uma comparação com o mencionado método SIFT. Foi medido o tempo que cada algoritmo levou para analisar as imagens, assim como a qualidade e a fidedignidade das semelhanças encontradas. A verificação da qualidade foi baseada na visualização dos resultados apresentados, após gerar a imagem final com as marcações das correspondências encontradas. A implementação do SIFT foi também na linguagem Python e executado na mesma máquina e com as mesmas imagens de entrada, a fim de construir um ambiente justo para a comparação.

V. Implementação

A versão estendida dos algoritmos implementados, assim como os *scripts* de execução dos testes, encontram-se num repositório do GitHub[2]. Os Códigos 2 e 3 são as implementações em Python dos algoritmos descritos na metodologia. Na prática, a única diferença do algoritmo descrito anteriormente é a obtenção da largura e altura das imagens que são essenciais para percorrer os *pixels* das imagens, tanto para criar os mapas, quanto para gerar a imagem final contendo as correspondências obtidas.

```
1 def lr2Matching(A, B):
2     width = A.width
3     height = A.height
4     gvA = grayvalues(A)
5     gvB = grayvalues(B)
6     mapA, ipA, ipB = selectPixels(gvA, gvB,
      width, height, 160)
7     mapB = selectPixelsBy(mapA,ipA,ipB,width)
8     return highlightPixels(A, B, mapA, mapB,
      gvA, gvB, 160)
```

Código 2: Implementação em Python do algoritmo com *grayvalues*

[1] https://github.com/Levysantiago/image-matching-algorithm/blob/master/docs/RelatorioParcial.pdf

[2] https://github.com/Levysantiago/image-matching-algorithm

```
def lr2MatchingRGB(A, B):
    width = A.width
    height = A.height
    pixelsA = pixels(A)
    pixelsB = pixels(B)
    mapA, ipA, ipB = selectPixelsRGB(pixelsA,
    pixelsB, width, height, 300)
    mapB = selectPixelsBy(mapA,ipA,ipB,width)
    return highlightPixelsRGB(A, B, mapA, mapB
    , pixelsA, pixelsB, 150)
```

Código 3: Implementação em Python do algoritmo com RGB

Também foi implementado o algoritmo SIFT para posterior comparação com os algoritmos propostos. O Código 4 mostra uma implementação proposta por [20]. Neste, percebe-se que a metodologia adota diversas etapas, incluindo a obtenção das imagens em formato preto e branco, a fim de utilizar os *grayvalues*, detecção de pontos principais da imagem (*keypoints*) e localização de *pixels*, cálculos de diferenças entre versões desfocadas da imagem, e, enfim, o encontro das correspondências e exibição do resultado.

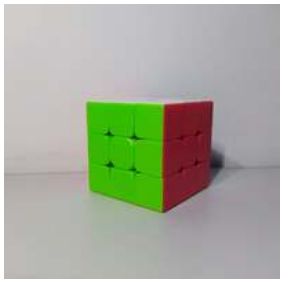
```
def SIFT(img1, img2, showImage=True, savePath=
    False):
    img1 = cv2.cvtColor(img1, cv2.
    COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(img2, cv2.
    COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints_1, descriptors_1 = sift.
    detectAndCompute(img1, None)
    keypoints_2, descriptors_2 = sift.
    detectAndCompute(img2, None)
    bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck
    =True)
    matches = bf.match(descriptors_1,
    descriptors_2)
    matches = sorted(matches, key=lambda x: x.
    distance)
    return cv2.drawMatches(img1,keypoints_1,
    img2,keypoints_2,matches[:50],img2,flags
    =2)
```

Código 4: Implementação em Python do algoritmo SIFT

VI. Análise e Discussão

Esta seção apresenta os resultados dos testes realizados com os algoritmos, respectivas complexidades e comparações de tempo e qualidade das correspondências geradas.

a) Hardware: Os testes foram executados numa máquina de processador Intel core i5, 8GB de memória RAM, placa de vídeo NVIDIA Geforce 940M. Porém, podem ser reproduzidos por qualquer outra a partir do código disponibilizado.



(a) Cubo posição inicial

(b) Cubo em outro ângulo

Figura 1: Imagem de objeto único

(a) Objetos posição inicial

(b) Objetos em outro ângulo

Figura 2: Imagem de vários objetos

b) Entrada: Para a comparação dos algoritmos, primeiramente, foram selecionadas algumas imagens, aplicando para cada versão uma transformação geométrica. Foram realizados cinco testes com fotos feitas por meio de um *smartphone* e redimensionadas para o tamanho 200X200 *pixels*. As Figuras 1 e 2 são exemplos de duplas de imagens utilizadas para aplicação dos testes. Na primeira imagem, em Figura 1a, o cubo mágico encontra-se em uma posição inicial, e na segunda, Figura 1b, está deslocado tanto de forma angular quanto horizontalmente. O mesmo acontece com as Figuras 2a e 2b, as quais foram propositalmente selecionadas a fim de testar se os algoritmos conseguem detectar correspondências entre vários objetos em imagens deslocadas.

c) Complexidade assintótica: Com relação ao algoritmo proposto, tanto utilizando *grayvalues*, quanto RGB, possui em geral complexidade $O(mn + k)$, onde m e n são a largura e a altura da imagem respectivamente. A fim de percorrer todos os *pixels* da primeira imagem na etapa de geração do seu mapa, o algoritmo faz um *loop* até mn. Como nos experimentos foram consideradas as imagens quadradas, seguindo a definição do problema (seção II), sendo o tamanho das imagens nxn, neste primeiro momento, a complexidade é $O(mn)$ ou $O(n^2)$. A etapa de criação do mapa da segunda imagem percorre somente os *pixels* selecionados da primeira imagem, sendo de complexidade $O(k_1)$, sendo k_1, o tamanho do mapa da primeira imagem. A etapa de destaque dos pontos correlacionados gera uma terceira imagem a partir da segunda, tendo complexidade $O(k_2)$, onde k_2 é o tamanho do mapa da segunda imagem. Tem-se que a complexidade geral é $O(n^2+k_1+k_2)$. Como os mapas possuem o mesmo tamanho,

sendo que o segundo mapa contém os *pixels* do primeiro mapa com um deslocamento aplicado, tem-se que $k_1 = k_2$, assim, pode-se chegar à complexidade $O(mn + k)$ ou $O(n^2 + k)$.

O algoritmo SIFT estabelece cinco etapas até o encontro da correspondência em si, e cada uma destas etapas apresentam uma complexidade diferente [21]. A detecção de extremos em escala espacial, apresenta uma complexidade $O(C_{krn}C_{oc}mn)$, onde C_{krn} é o tamanho da matriz de convolução desta técnica para ofuscar as imagens, C_{oc} é o número de escalas geradas. A etapa de localização de *keypoints*, onde cada *pixel* é comparado com seus vizinhos, a complexidade é dada pela Equação 2. Depois de encontrar os *keypoints*, cada um deles é analisado a fim de calcular as suas exatas localizações, esta etapa tem complexidade $O(k) + O(k) = O(k)$, onde k é o número de extremos encontrados na etapa anterior. A penúltima etapa consiste em associar uma orientação para cada *keypoint*, baseando-se nas localizações, definindo descritores relacionados à esta orientação, esta etapa também tem complexidade $O(k) + O(k) = O(k)$. A última etapa consiste no encontro da correspondência em si, considerando todas as saídas das etapas anteriores. Utilizando uma árvore binária balanceada com profundidade igual a $log_2 k_1$, o algoritmo separa os *keypoints* em grupos similares para ajudar no encontro das correspondências, todo o processo de encontro da correspondência tem complexidade $O(k_2 log(k_1))$, onde k_1 é o número de *keypoints* da primeira imagem e k_2 o número de *keypoints* da segunda imagem.

$$T(m,n) = O(mn - \frac{mn}{min(m,n)}) \tag{2}$$

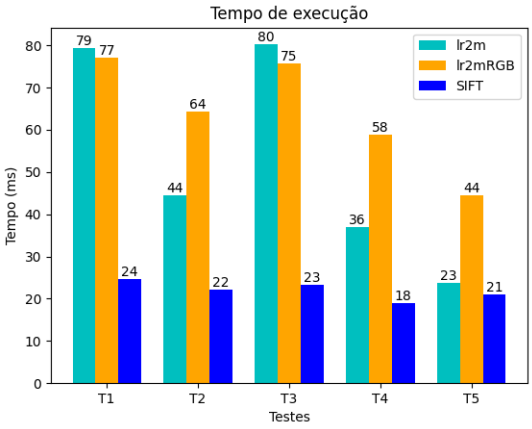
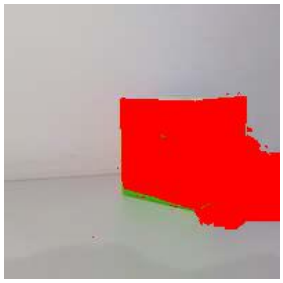
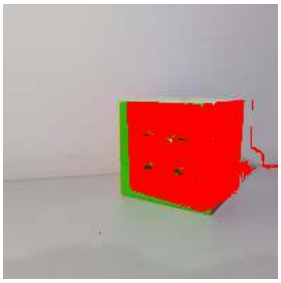


Figura 3: Gráfico comparativo do tempo de execução dos algoritmos em cada teste

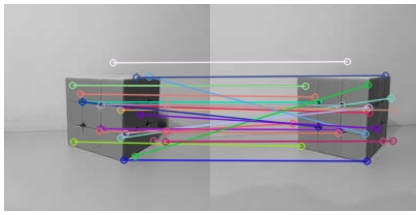
d) Tempo de execução: Para os testes: T1, T2, T3, T4 e T5, foi medido o tempo de execução de cada um dos três algoritmos: lr2m, lr2mRGB e SIFT. A Figura 3 apresenta um gráfico de barras dos resultados da execução, estando representados no eixo X os mencionados testes e no eixo Y o tempo da execução em milissegundos (ms). Diante dos resultados, percebe-se uma grande variação nos tempos de resposta dos algoritmos propostos: lr2m e lr2mRGB, pois ambos



(a) Teste T2 com lr2m



(b) Teste T2 com lr2mRGB



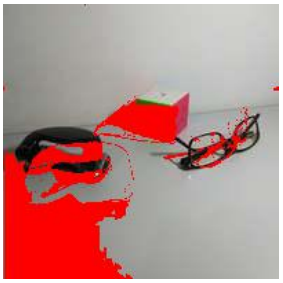
(c) Teste T2 com SIFT

Figura 4: Resultados teste T2

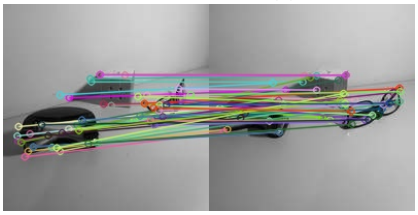
dependem de quantos *pixels* foram selecionados para os mapas e quantos *pixels* devem ser destacados. Embora o lr2m utilize *pixels* em *grayvalues* e o lr2mRGB, *pixels* em RGB, ambos os algoritmos propostos apresentaram tempos de resposta muito próximos para T1 e T3. Nos outros testes, porém, observou-se uma diferença a favor do lr2m com relação ao lr2RGB. Em todos os testes, é notório o melhor desempenho do SIFT se comparado ao lr2m e ao lr2mRGB. Como explicado anteriormente, o SIFT inclui diversas técnicas consolidadas, as quais influenciaram fortemente no seu melhor desempenho. Outro fato interessante a ser levado em consideração é a constância nos resultados obtidos por este último algoritmo, que apresentou uma variação muito pequena em cada caso de teste. Isso pode estar relacionado ao tamanho constante da entrada ser sempre 200X200.

e) Qualidade das correspondências: Foram incluídas nesta etapa as correspondências entre os *pixels* encontradas por cada um dos três algoritmos com relação aos testes T1 e T2. As Figuras 4 e 5 ilustram as mencionadas correspondências. Os *pixels* encontrados pelos algoritmos propostos estão na cor vermelha, enquanto os *pixels* encontrados pelo SIFT estão ligados por linhas entre as duas imagens. No T2, percebe-se que o lr2m encontrou as correspondências do cubo na Figura 4a. Porém, selecionou também *pixels* na área de sombra do objeto. Na Figura 4b, observa-se que o lrm2RGB identificou melhor os *pixels* do objeto porque as suas cores são bem diferente do *background*. Porém, sendo o fundo de cor mais escura, o lrm2RGB pode gerar uma confusão na seleção das correspondências. Na Figura 4c, o SIFT apresentou mais claramente a relação entre as duas imagens, de forma a deixar evidente os exatos pontos de correspondência entre elas. No T3 (Figura 5), percebe-se que o lr2m cometeu o mesmo ato de selecionar *pixels* relacionados ao sombreamento dos objetos, o que não é um erro, mas é sempre interessante filtrar da melhor forma possível as correlações dos objetos que estão

(a) Teste T3 com lr2m



(b) Teste T3 com lr2mRGB



(c) Teste T3 com SIFT

Figura 5: Resultados teste T3

na imagem do que as sombras, o que pode atrapalhar numa posterior observação dos resultados. O algoritmo lr2mRGB, assim como no teste T2, seleciona um pouco melhor os *pixels* correspondentes dos objetos, mas seleciona também alguns *pixels* da área de sombra, como pode ser visto na Figura 5b, o que, como dito, não é bem um erro, mas dependendo da situação, pode ser interessante desconsiderar o que seja irrelevante. Por sua vez, o SIFT que aplica melhores técnicas, selecionou os *pixels* exatos de correspondência com cada objeto.

VII. Conclusão e perspectivas

A *Image Matching* é um problema muito conhecido, que visa à identificação da correspondência de estrutura/conteúdo igual ou semelhante de duas imagens. Semelhante a uma foto feita de ângulos diferentes de um mesmo objeto, a segunda imagem é uma transformação geométrica da primeira. Este problema está presente nos diversos campos, a exemplo de navegação, orientação, vigilância automática, fotogrametria, visão robótica e medicina. Atualmente, existem diversos métodos e algoritmos que propõem resolver o problema de diferentes formas, como por exemplo, o SIFT, que, aplicando diversas etapas de localização de *pixels* vizinhos e encontro de pontos chave, consegue identificar de forma eficiente a correspondência entre as imagens. Este trabalho propôs uma nova metodologia e implementação para resolver tal problema, sem a pretensão de ser melhor ou mais rápida do que as soluções existentes, mas que demonstre um novo modo de pensar na resolução do problema. Este trabalho implementa o algoritmo proposto em duas versões, a saber, lr2m e lr2mRGB. Segue analisando comparativamente os resultados das implementações propostas com os resultados da implementação consagrada SIFT, focando no tempo de execução e qualidade das correspondências. O algoritmo lr2mRGB apresentou uma maior qualidade do que o lr2m, porém foi mais lento. Em geral, os mencionados algoritmos resolvem o problema de forma limitada e com uma qualidade menor que o SIFT. Este, por sua vez, resolve os problemas mais rapidamente. Em trabalhos futuros, pretende-se melhorar a qualidade das correspondências geradas pelos algoritmos, por meio da aplicação de métricas amplamente divulgadas na literatura, visando a redução do tempo de execução e aumento da *performance*. Também, a seleção dos *pixels* da área de sombreamento aponta para uma necessidade de melhoria e aperfeiçoamento futuro dos algoritmos propostos.

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[2] D. Keysers and W. Unger, "Elastic image matching is np-complete," *Pattern Recognition Letters*, vol. 24, no. 1-3, pp. 445–453, 2003.

[3] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, "Image matching from handcrafted to deep features: A survey," *International Journal of Computer Vision*, pp. 1–57, 2020.

[4] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] B. Huet and E. R. Hancock, "Structural indexing of infra-red images using statistical histogram comparison," in *Proceedings IWISP'96*. Elsevier, 1996, pp. 653–656.

[7] S.-L. Hsieh, C.-C. Chen, and C.-R. Chen, "A novel approach to detecting duplicate images using multiple hash tables," *Multimedia Tools and Applications*, vol. 74, no. 13, pp. 4947–4964, 2015.

[8] Y. Amit, "A nonlinear variational problem for image matching," *SIAM Journal on Scientific Computing*, vol. 15, no. 1, pp. 207–224, 1994.

[9] P. Dupuis, U. Grenander, and M. I. Miller, "Variational problems on flows of diffeomorphisms for image matching," *Quarterly of applied mathematics*, pp. 587–600, 1998.

[10] R. A. Paringer, Y. Donon, and A. V. Kupriyanov, "Modification of blurred image matching method," *Computer Optics*, vol. 44, no. 3, pp. 441–445, 2020.

[11] G. Hermosillo, C. Chefd'Hotel, and O. Faugeras, "Variational methods for multimodal image matching," *International Journal of Computer Vision*, vol. 50, no. 3, pp. 329–343, 2002.

[12] A. Gruen, "Adaptive least squares correlation: a powerful image matching technique," *South African Journal of Photogrammetry, Remote Sensing and Cartography*, vol. 14, no. 3, pp. 175–187, 1985.

[13] P. A. Van den Elsen, E.-J. Pol, and M. A. Viergever, "Medical image matching-a review with classification," *IEEE Engineering in Medicine and Biology Magazine*, vol. 12, no. 1, pp. 26–39, 1993.

[14] S. Uchida and H. Sakoe, "A monotonic and continuous two-dimensional warping based on dynamic programming," in *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No. 98EX170)*, vol. 1. IEEE, 1998, pp. 521–524.

[15] F. A. W. Belo, "Desenvolvimento de algoritmos de exploração e mapeamento visual para robôs móveis de baixo custo," *Rio de Janeiro*, 2006.

[16] R. C. Gonzales and R. E. Woods, "Digital image processing," 2002.

[17] S. Kapur, *Computer Vision with Python 3*. Packt Publishing Ltd, 2017.

[18] J. E. Solem, *Programming Computer Vision with Python: Tools and algorithms for analyzing images*. "O'Reilly Media, Inc.", 2012.

[19] S. Dey, *Hands-On Image Processing with Python: Expert techniques for advanced image analysis and effective interpretation of image data*. Packt Publishing Ltd, 2018.

[20] A. Singh, "A detailed guide to the powerful sift technique for image matching (with python code)," https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/, 2019, [Acesso em: 11 Dez. 2020.].

[21] P. Drews, R. de Bem, and A. de Melo, "Analyzing and exploring feature detectors in images," in *2011 9th IEEE International Conference on Industrial Informatics*. IEEE, 2011, pp. 305–310.