Universidade Federal do Rio Grande do Norte

Programa de Pós-graduação em Engenharia Elétrica e de Computação

EEC1515 - VISÃO COMPUTACIONAL

RELATÓRIO

IMPLEMENTAÇÃO DE UM DETECTOR DE QUADRADOS BASEADO NA TRANSFORMADA HOUGH

Autor: Luís Gabriel Pereira Condados

Professor orientador: Rafael Beserra Gomes

Natal-RN

2020

Universidade Federal do Rio Grande do Norte

Programa de Pós-graduação em Engenharia Elétrica e de Computação

EEC1515 - VISÃO COMPUTACIONAL

RELATÓRIO

Relatório apresentado à disciplina de EEC1515- Visão Computacional, correspondente a 1º unidade do semestre 2020.2, sob orientação do **Prof. Rafael Beserra Gomes**.

Autor:Luís Gabriel Pereira Condados

Natal-RN

2020

Sumário

Lista de ilustrações

1 INTRODUÇÃO

A transformada de *Hough* é um método comum para detecção de formas que são facilmente parametrizáveis (formas comuns: linhas e círculos). Geralmente essa transformada é utilizada após a etapa de pré-processamento da imagem, principalmente após detecção de bordas.

O método consiste em mapear determinados pixeis da imagem numa determinada célula no espaço de parâmetros que definem a figura geométrica. Esse procedimento é repetido para toda a imagem e as células do espaço de parâmetros são incrementadas, servindo assim de indicadores da existência de uma determinada forma.

Neste trabalho será apresentado uma implementação da transformada *Hough* para a identificação de quadrados pretos em imagens com fundo branco baseado no gradiente. Para isso foi feito a parametrização de um quadrado com a informação do centro do quadrado, tamanho do lado e orientação com relação ao eixo horizontal (x_c, y_c, l, θ), por isso o espaço de configuração terá dimensão 4. Para fazer o mapeamento entre o espaço de imagem e o de parâmetros foi utilizado a informação do gradiente da imagem e um ponto de borda, para com isso obter-se as normais do quadrado e com isso estimar os quatro parâmetros. Após o mapeamento/acumulo dos indicadores é feito uma etapa de filtragem dos quadrados, para melhorar a precisão da detecção.

2 METODOLOGIA

Como já mencionado, o objetivo deste trabalho foi apresentar e testar uma abordagem de identificação de quadrados pretos em imagens brancas, fazendo uso da transformada *Hough* e do gradiente da imagem. A parametrização dos quadrados utilizada foi: centroide (x_c, y_c); orientação com relação ao eixo horizontal (θ) e o comprimento de seu lado (l) em pixeis (a Figura 1 ilustra essa parametrização).

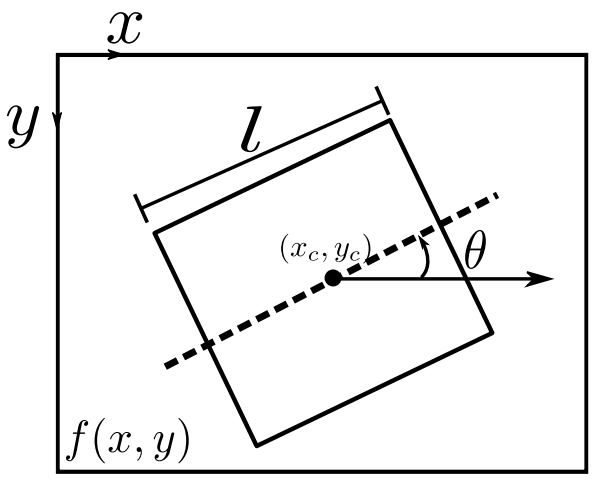


Figura 1 – Parametrização do quadrado em uma imagem $f(x, y)$.

Na Figura 1 podemos ver uma ilustração de um quadrado (não preenchido) em uma imagem, referenciada por $f(x, y)$, com os parâmetros que o definem sendo destacados. Também é importante notar nessa imagem que o sistema de coordenadas utilizado é localizado no canto superior esquerdo da imagem e sua orientação também é como o ilustrado.

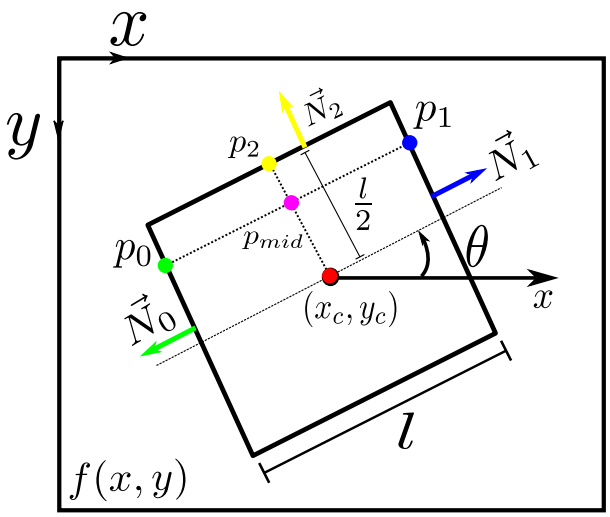


Figura 2 – Identificação dos parâmetros do candidato à quadrado.

A Figura 2 apresenta pontos e vetores chaves para o mapeamento em espaço de parâmetros, as notações presentes nessa figura será utilizado nas descrições a seguir.

A primeira etapa do algoritmo proposto é calcular o gradiente $\nabla f(x, y)$ da imagem de entrada $f(x, y)$. O módulo dele $|\nabla f(x, y)|$ será utilizado para indicar quais pixeis fazem parte de alguma borda, para isso foi definido um limiar que se ultrapassado sinalizará que o mesmo pertence a uma borda. O limiar utilizado neste trabalho foi escolhido empiricamente, o valor utilizado em todos os experimentos foi de 200, porém vale salientar que devido a imagem de entrada possuir apenas pixeis 100% pretos ou 100% brancos e a uma etapa final de filtragem de quadrados no método proposto (que será explicado mais adiante), o limiar possuí uma ampla faixa de valores que permitem o bom funcionamento do algoritmo.

A proposta consiste em visitar cada pixel em $f(x, y)$ e analisar o módulo do gradiente nesse ponto, caso seja classificado como borda, usa-se a informação do gradiente, que corresponde ao vetor normal do lado ao qual esse pixel pertence (desconsiderando os erros de quantização devido à natureza digital da imagem), utilizando a informação do gradiente nesse ponto é estimado a orientação do quadrado fazendo $\vec{N}_0 \mod 90°$ (isso é possível devido à simetria de um quadrado). Note que é possível também, estimar as demais normais do quadrado a partir de $\vec{N}_0$, também fazendo-se uso da simetria da figura. As demais normais servirão para facilitar a compreensão em qual sentido deve-se buscar os demais pontos. A partir de um ponto p_0 de borda identificado, percorre-se no sentido

de $-\vec{N_0}$ (ou $\vec{N_1}$)(ver Figura 1) até encontrar um segundo ponto de borda p_1. A distância entre ambos os pontos indicara o comprimento l do lado do quadrado.

A próxima etapa é caminhar em uma direção ortogonal ao vetor $\overrightarrow{p_0p_1}$ (uma maneira de obter esse vetor é rotacionar N_0 em 90°, como foi feito neste trabalho) até encontrar um terceiro ponto de borda p_2. Com p_2 e a estimativa do tamanho l, estima-se o centro do quadrado fazendo: $(x_c, y_c) = p_2 - \vec{N_2}\frac{l}{2}$, sendo $\vec{N_2}$ o vetor normal no ponto p_2. Com esse procedimento é feito o mapeamento do domínio da imagem para o de parâmetros de um quadrado: $(x, y) \Rightarrow (x_c, y_c, l, \theta)$ e é realizado um voto na matriz de votação $(M[x_c][y_c][l][\theta] = M[x_c][y_c][l][\theta] + 1)$.

A seguir é apresentado um pseudo código ilustrando o procedimento de mapeamento $(x, y) \Rightarrow (x_c, y_c, l, \theta)$.

Algorithm 1 A Hough Transform for Squares Detection

initialize M with zero. ▷ Voting matrix initialized with zero.
for all (x', y') in $f(x, y)$ **do**
  if $|\nabla f(x', y')| \geq$ edge threshold **then** ▷ It's an edge.
    $p_0 \leftarrow (x', y')$
    $\theta \leftarrow \angle\nabla f(x', y') \mod 90°$ ▷ Square's angle.
    $\vec{N_0} \leftarrow \nabla f(x', y')/|\nabla f(x', y')|$
    $\vec{N_1} \leftarrow -\nabla f(x', y')/|\nabla f(x', y')|$
    $\vec{N_2} \leftarrow$ rotate $\vec{N_1}$ on 90°

    Search for an edge point going in the direction of $\vec{N_1}$ from p_0. ▷ That will be the p_1 point.

    $l \leftarrow |p_0 - p_1|$ ▷ Square's size.
    $p_{middle} \leftarrow (p_1 + p_0)/2$

    Search for an edge point going in the direction of $\vec{N_2}$ from p_{middle}. ▷ That will be the p_2 point.

    $p_{center} \leftarrow p_2 - \vec{N_2} * l/2$ ▷ $p_{center} = (x_c, y_c)$
    $M[x_c][y_c][l][\theta] \leftarrow M[x_c][y_c][l][\theta] + 1$
  end if
end for

Uma vez que tenha-se percorrido por completo o domínio de $f(x, y)$, faz-se a seleção de quais conjuntos de parâmetros na matriz M pode ser considerado um quadrado, para isso é definido outro limiar, esse por sua vez indica a partir de quantos votos um dado candidato a quadrado pode ser considerado de fato um quadrado presente na imagem de entrada. No presente trabalho esse limiar foi usado como sendo igual a 4.

É realizado uma etapa final de filtragem/supressão na lista de quadrados identificados, objetivando melhorar a identificação e para tornar o algoritmo menos sensível à escolha do limiar de votos mínimos. Percorre-se a lista que contém todos os supostos quadrados, que foram classificados como tal devido ao processo de contagem da contagem dos votos, e busca-se grupos destes que apresentem superposições ou interseções entre si, ao ser identificado essas situações é criado um quadrado representante a partir da media dos parâmetros dos candidato à quadrado que se encontram em superposição. Ou seja, é calculado o quadrado médio em caso de superposição de quadrados. A identificação da superposição é feita analisando a distancia entre os centros dos quadrados, caso essa distância seja menor que a soma da metade das diagonais dos quadrados, será considerado uma superposição e faz-se a média.

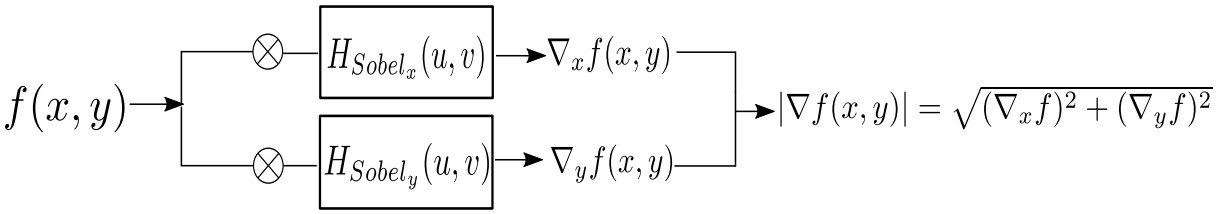


Figura 3 – Cálculo do módulo do gradiente da imagem $f(x, y)$.

A Figura 2 ilustra o procedimento para o cálculo do módulo do gradiente da imagem. Obtém-se inicialmente as componentes do gradiente nas direções x e y, por meio da convolução de $f(x, y)$ com os operadores de *Sobel* e por fim calcula-se o módulo.

$$H_{Sobel_x} = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

(a) Operador de *Sobel* na direção x.

$$H_{Sobel_y} = \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

(b) Operador de *Sobel* na direção y.

Figura 4 – Operadores de *Sobel*.

Foi utilizado a convolução entre a imagem ($f(x, y)$) com os operadores de *Sobel* para obter-se o gradiente da imagem. A Figura 4 apresenta esses operadores. Aqui também é importante notar que ambos os operadores (Figuras 4a e 4b) estão de acordo com o sistema de coordenadas adotadas, ou seja, eixo x crescente para a direita e y crescente para baixo, considerando o referencial limite superior esquerdo da imagem, como já mencionado.

Visando a validação do trabalho foi feita uma implementação (em *C++* e fez-se uso da biblioteca de visão computacional *OpenCV*(1)) do procedimento proposto. Aplicando diferentes imagens com fundo branco e com um número conhecido de quadrados pretos,

como entrada e gerando como saída uma imagem contendo os quadrados identificados em vermelho superpondo a imagem de entrada. Dessa forma é fácil analisar visualmente se todos os quadrados foram identificados corretamente, além disso o programa exibe uma saída no terminal com a contagem do número de quadrados identificados antes e depois da etapa de filtragem. Algumas imagens extras são geradas também, como a de gradiente e uma imagem contendo os quadrados identificados na etapa pré filtragem, apenas para fins comparativos.

O programa contém alguns parâmetros além dos já mencionados (quantidade de votos mínimos e limiar para classificação da borda) que podem ser ajustáveis diretamente no código, tais parâmetros servem para controlar o quanto de memoria pode será utilizado, pois por se tratar de uma matriz de parâmetros quadridimensional, mesmo usando-se de palavras pequenas (foi utilizado variáveis de 16bits de tamanho para ser os acumuladores) facilmente o programa exigirá grandes quantidades de memoria. Por exemplo:

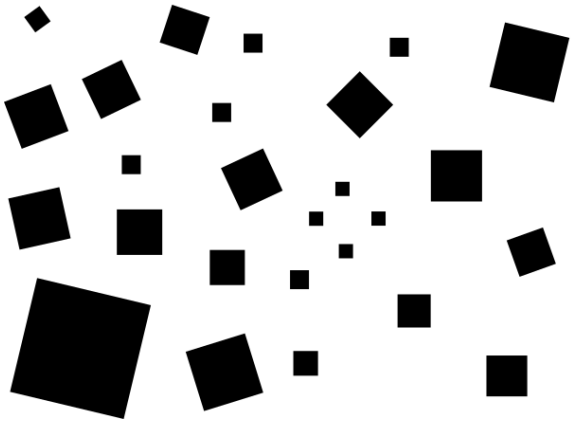
Para imagens de 640 x 480 pixeis e com os lados dos quadrados variando de 2 até 480 pixeis e realizando uma discretização dos ângulos de 0 à 90 com 90 pontos igualmente espaçados. Usando-se palavras de 16 bits (2 bytes), a memoria necessária para alocar a matriz de parâmetros será de aproximadamente:

$$memory\_needed = 640 * 480 * (480 - 2) * 90 * 2 * 10^{-9} \approx 26.43Gb$$

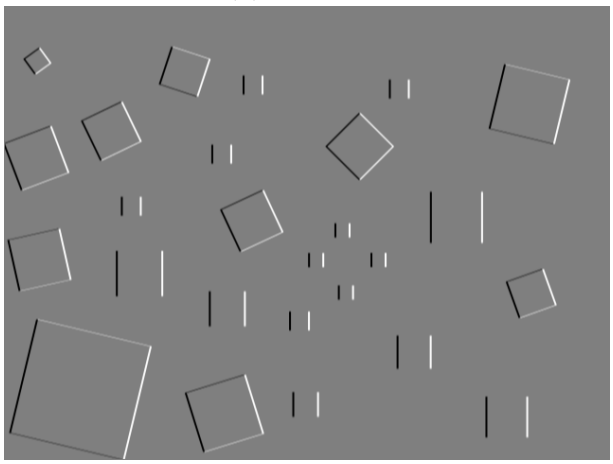
Os parâmetros extras são: tamanho mínimo e máximo de lado de quadrados que serão identificados e a discretização do ângulo. A discretização do ângulo foi usada, na maioria dos experimentos, igual à $90/5 = 18$. O que se traduz para uma precisão de $5°$ graus. Já o tamanho dos lados dos quadrados foi para a maioria dos experimentos: $l_{min} = 5$ e $l_{max} = 100$ px.

3 RESULTADOS

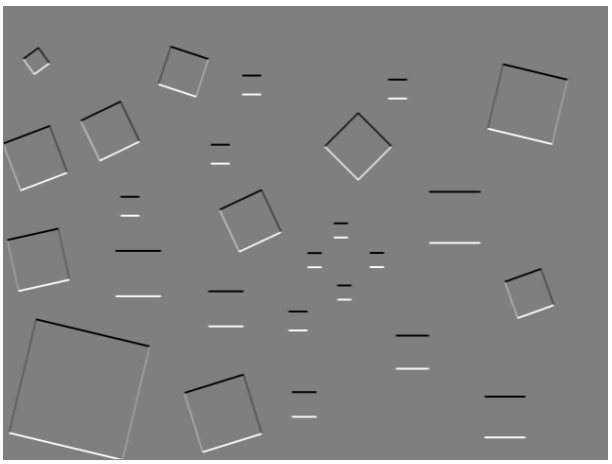
Aqui serão apresentados os resultados de 5 experimentos diferentes, ou seja, 5 imagens de entrada diferentes, em todos eles foi utilizado os parâmetros como mencionados na seção de metodologia. Para cada experimento é apresentado um grupo de 5 imagens, sendo elas: A imagem de entrada; as duas imagens de gradiente da imagem, na direção x e y (vale saliente que para fins de exibição esses gradientes foram mapeados de [-1020,1020] para [0,255], fazendo com que as cores em branco representem a componente do gradiente apontando na direção crescente do eixo em questão; pixeis pretos representam o contrário e cinzas representam gradiente zero.); imagem de entrada sobrepostas com quadrados com bordas vermelhas representando os quadrados identificados antes da etapa de filtragem/supressão e por fim a imagem final, pós filtragem.



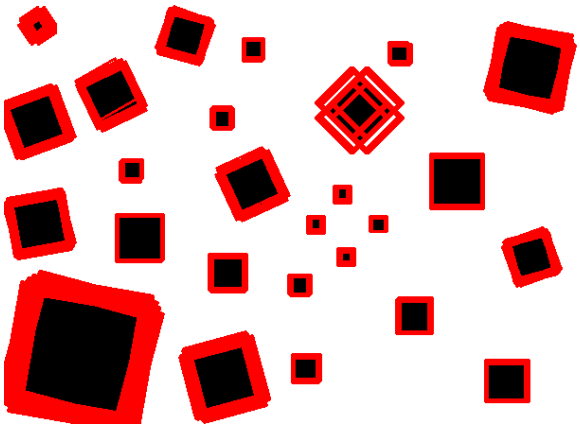
(a) Entrada.



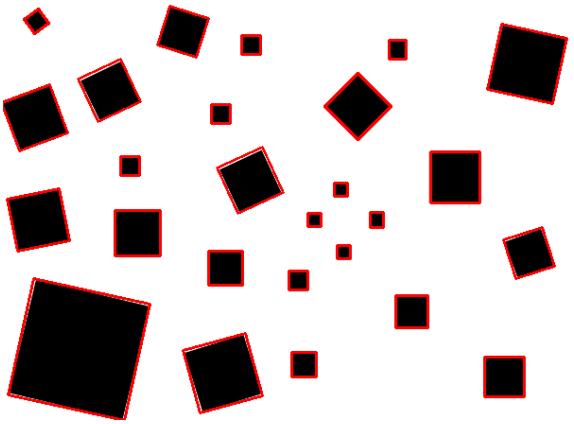
(b) Gradiente na direção x.



(c) Gradiente na direção y.

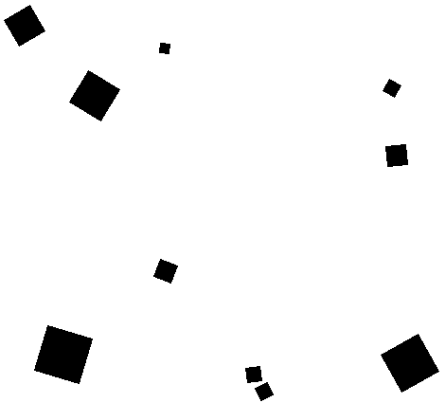


(d) Saída não filtrada.

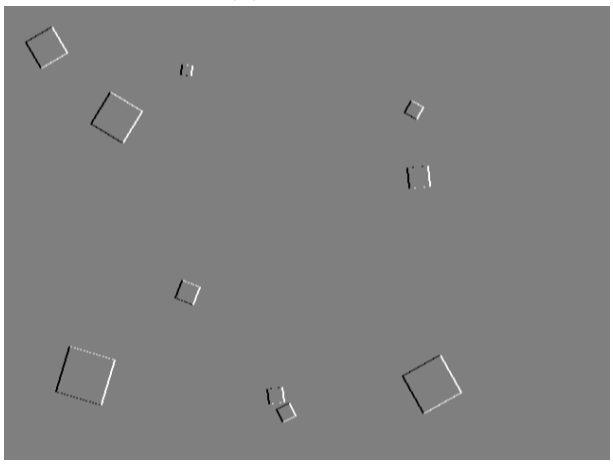


(e) Saída filtrada.

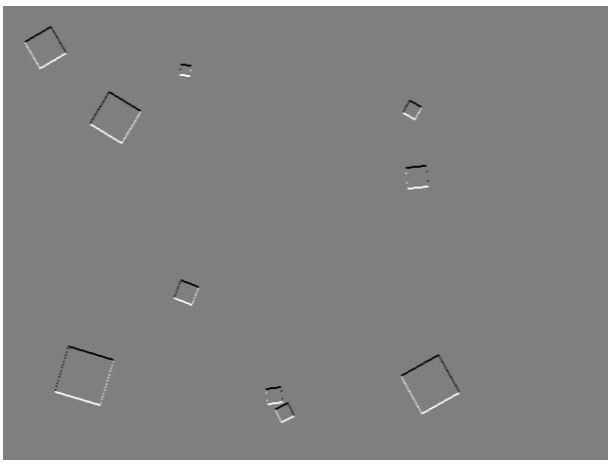
Figura 5 – Experimento 1. Imagem com 26 quadrados de diferentes tamanhos e diferentes orientações.



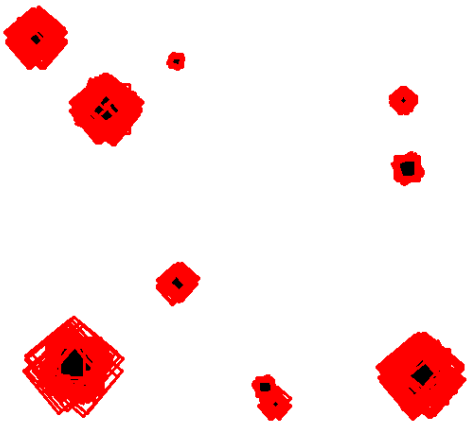
(a) Entrada.



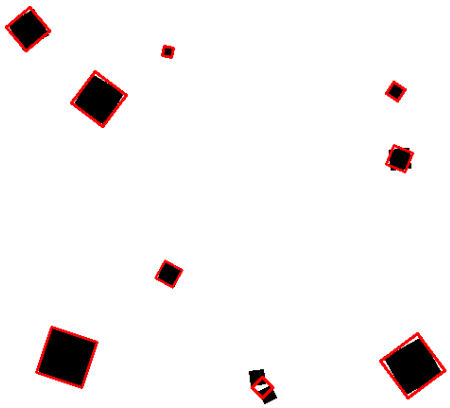
(b) Gradiente na direção x.



(c) Gradiente na direção y.

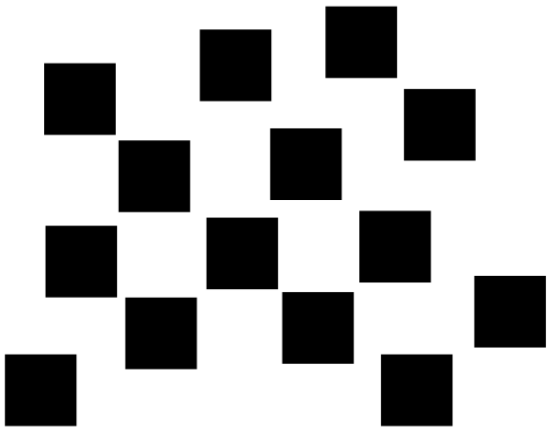


(d) Saída não filtrada.

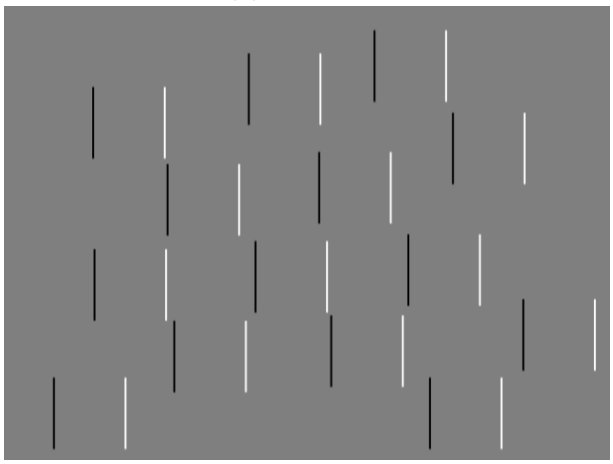


(e) Saída filtrada.

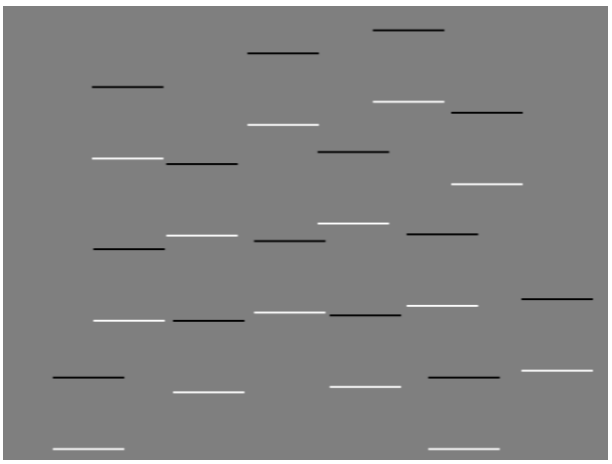
Figura 6 – Experimento 2. Imagem com 10 quadrados de diferentes tamanhos e diferentes orientações.



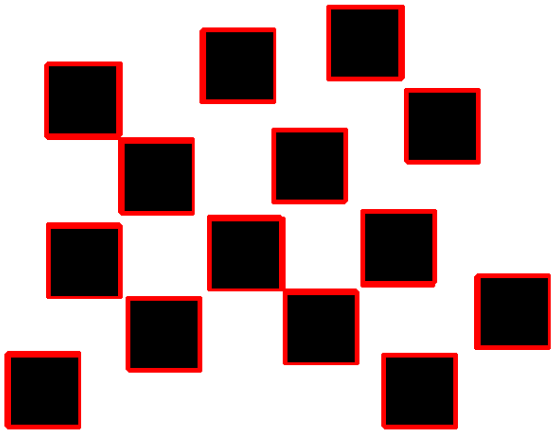
(a) Entrada.



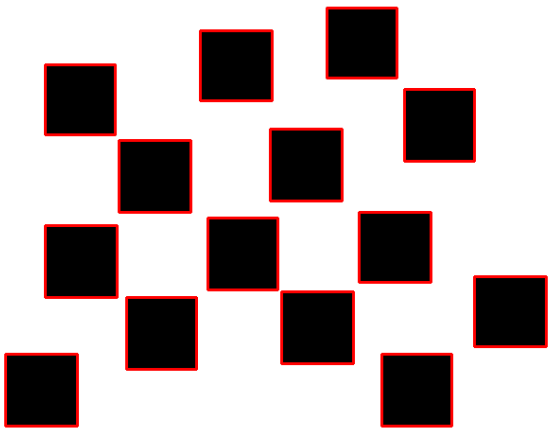
(b) Gradiente na direção x.



(c) Gradiente na direção y.

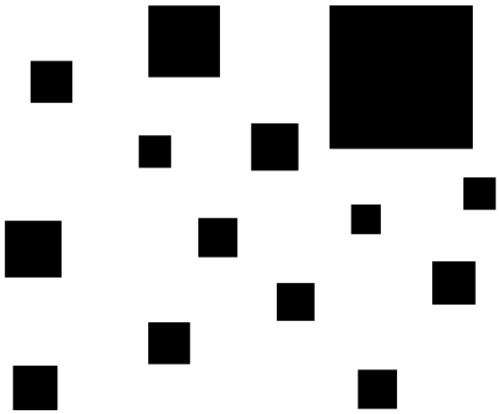


(d) Saída não filtrada.

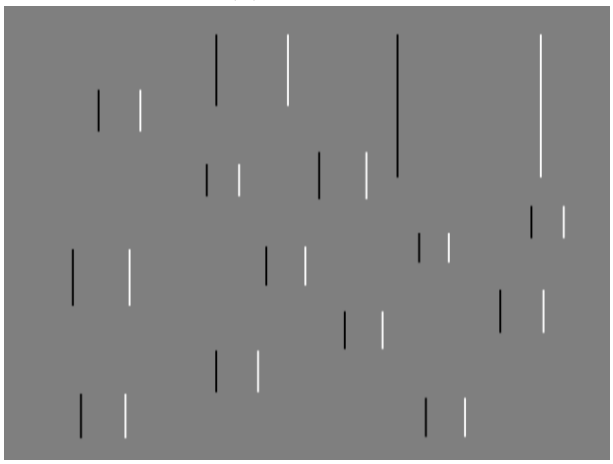


(e) Saída filtrada.

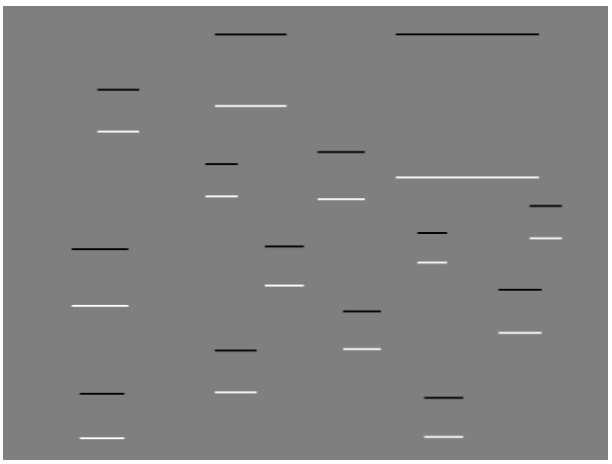
Figura 7 – Experimento 3. Imagem com 14 quadrados de tamanhos e orientações iguais.



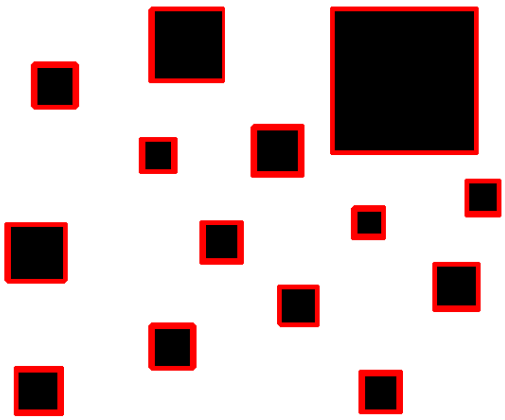
(a) Entrada.



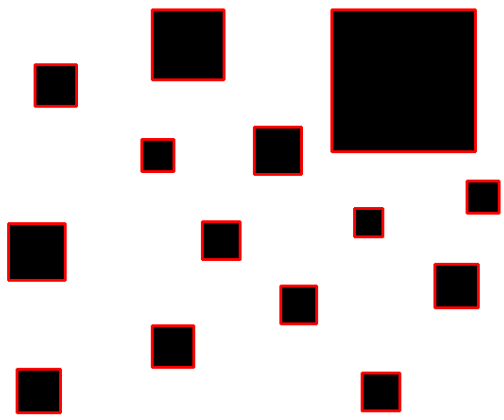
(b) Gradiente na direção x.



(c) Gradiente na direção y.

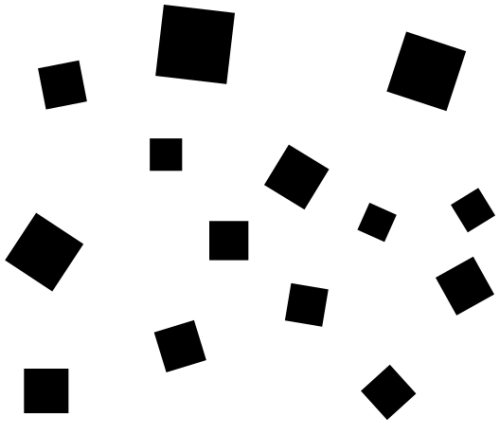


(d) Saída não filtrada.

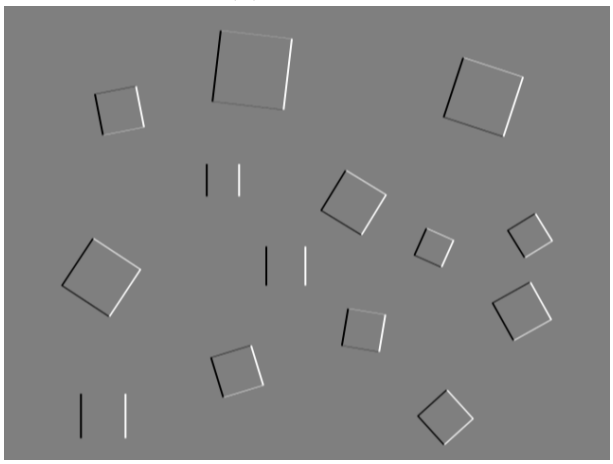


(e) Saída filtrada.

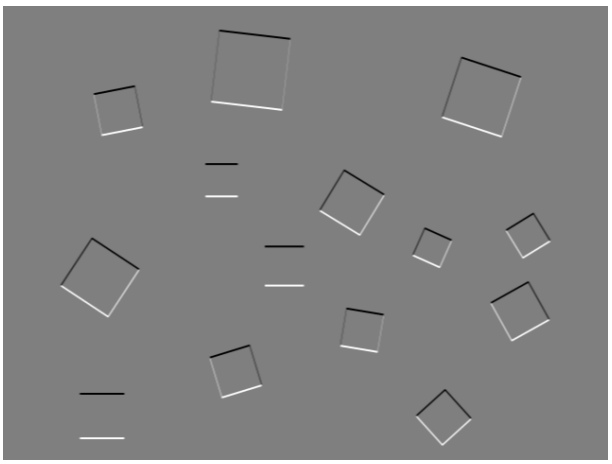
Figura 8 – Experimento 4. Imagem com 14 quadrados com tamanhos diferentes e orientações iguais.



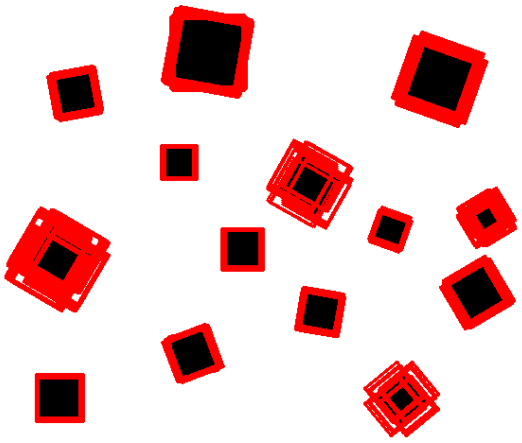
(a) Entrada.



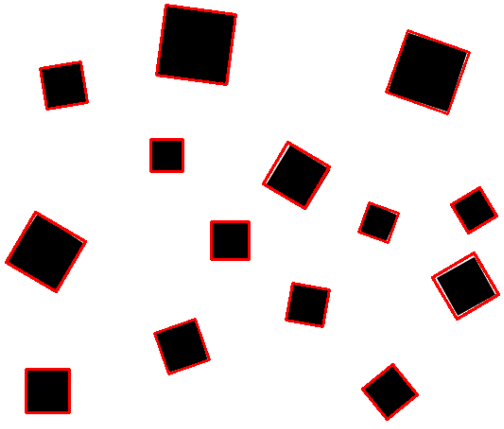
(b) Gradiente na direção x.



(c) Gradiente na direção y.



(d) Saída não filtrada.



(e) Saída filtrada.

Figura 9 – Experimento 5. Imagem com 14 quadrados com tamanhos diferentes e orientações diferentes.

No experimento 1 (Figura 5) a imagem de entrada contem 26 quadrados em diferentes posições e com diferentes tamanhos e orientações. Todos os 26 quadrados foram identificados na imagem final nesse experimento. Na saída sem supressão foram identificados mais de 300 quadrados. A imagem de entrada deste experimento foi gerada por software de desenho.

O experimento 2 (Figura 6) teve como entrada uma imagem com 10 quadrados diferentes. Foram identificados 9 na imagem de saída (pós supressão) e mais de 90 sem a supressão. Na imagem de entrada os quadrados pretos foram gerados usando-se uma função de preenchimento de polígonos do *OpenCV*.

A imagem de entrada no experimento 3 possuí 14 quadrados com a mesma orientação e tamanho. Foram identificados 14 quadros na imagem final, como ilustrado na Figura 7. 152 quadrados foram identificados antes da supressão. A imagem de entrada foi gerada usando-se um software de desenho. O mesmo resultado final foi obtidos para os experimentos 4 e 5 (ver Figuras 8 e 9), sendo que nesses a imagem de entrada possui diferentes tamanhos (experimento 4) e orientações (experimento 5). O número de quadrados identificados antes da filtragem foram de 152 e 170 para os experimentos 4 e 5 respectivamente.

4 CONCLUSÃO

Observa-se que no experimento 3 (Figura 7) o procedimento falhou ao reconhecer um quadrado entre dois quadrados pequenos que estão muitos próximos. Isso foi devido ao procedimento de filtragem/supressão de quadros que supostamente deveriam estar em situação de superposição, podemos concluir isso, pois na imagem pré-filtragem os mesmos encontram-se separados, embora, muito próximos. Isso pode ter sido causado por conta da quantização nas coordenadas dos centros de ambos os quadrados, fazendo com que o teste de superposição interpreta-se essa situação de proximidade como sendo a superposição.

Uma outra observação que podemos fazer com relação a esse mesmo experimento, é a qualidade das imagens de gradiente. Podemos notar que se comparados as demais esse experimentou possui a imagem de gradiente com a pior qualidade, o que pode ter influenciado no resultado ruído na imagem pré filtragem, uma vez que o procedimento proposto é fortemente sensível ao gradiente.

Os demais experimentos apresentaram uma correta identificação de todos os quadrados presentes na imagem. Embora todos tenham tido um elevado número de identificação (identificações redundantes e/ou ruidosas) na imagem de pré filtragem. Isso é devido ao limiar que indica o número mínimos de votos ser baixo. Porém as imagens finais apresentaram-se sem as redundâncias.

Podemos concluir, por tanto, que pelos resultados apresentados anteriormente o procedimento proposto apresenta-se como uma boa técnica para a identificação de quadrados pretos usando-se a transformada *Hough*. Sendo capaz de identificar os quadrados em suas diferentes variações de localização, tamanho e orientação.

Referências

1 ITSEEZ. *Open Source Computer Vision Library*. 2015. <https://github.com/itseez/opencv>. 4

5 ANEXO 1

Código fonte: <https://github.com/Gabriellgpc/ComputerVision_DIM0141/blob/master/unidade2/tarefa3/2.cpp>