Algorithm for Locating the Vertices of a QR Code and Removing Perspective

Heitor Eugênio Gonçalves, Luciano Xavier Medeiros and Alexandre Coutinho Mateus

***Abstract*—Scanning QR Codes from cellular phone cameras has made reading this type of two-dimensional code more accessible. However, QR Code photos may show geometric distortions caused by camera positioning in relation to the image to be read and thereby, resulting in a perspective image. These deformations make it difficult to decode QR Codes and thinking about it, this article proposes an algorithm that, first, finds the vertices of the polygon that delimits the area of a QR Code and from the coordinates of these vertices, the algorithm removes the perspective from the image.**

***Index Terms*—QR Code, perspective, image correction, image recognition, image processing.**

I. Introdução

O Quick Response Code (QR Code) é um código bidimensional de rápida leitura e com grande capacidade de armazenamento, sendo capaz de codificar dados numéricos, alfanuméricos, binários e kanji [1, 2]. A princípio, o QR Code, desenvolvido em 1994 pela Denso-Wave Incorporated, era aplicado na indústria automobilística com o intuito de classificar componentos de veículos durante o processo de fabricação [3].

Os QR Codes têm numerosas aplicações, sendo usados na divulgação de endereço de *websites*, na medicina [4], em atividades de ensino [5], na identificação de usuários [6], na determinação do deslocamento de robôs [7], dentre outras funcionalidades. Essa grande empregabilidade do QR Code deve-se a facilidade de decodificá-lo com câmeras de celular, tornando-o muitíssimo acessível a pessoas comuns.

Devido as diversas utilidades dos QR Codes, a indústria de aplicativos tem investido no desenvolvimento de softwares capazes de ler esses códigos por meio de imagens obtidas por câmeras de celulares. Contudo, dependendo da perspectiva da imagem fotografada, o resultado é um QR Code que não é quadrado e assim, os módulos presentes na imagem possuem deformações geométricas. Tais distorções, frequentemente, impedem a decodificação dos dados presentes nas imagens desse tipo de código.

Em Tribak e Zaz [8], os *finder patterns* e os *aligment patterns* são identificados para que possa ser feito a correção do formato do QR Code. Nesse trabalho, uma varredura horizontal e vertical são realizadas para localizar preliminarmente os padrões de código QR, seguido pelo método Análise da Componente Principal que permite a remoção de falsos positivos.

Em um outro trabalho de Tribak e Zaz [9], identifica-se a utilização dos 7 momentos invariantes de Hu para filtrar o conjunto de candidatos a *finder pattern* obtidos preliminarmente. Essas regiões são usadas para determinar a distorção apresentada pelo QR Code.

No trabalho de Belussi e Hirata [10], o posicionamento do QR Code é identificado com rapidez em imagens com fundos arbitrários. Isso é feito a partir da localização dos *finder patterns* por meio do treinamento de classificadores e a execução de uma classificação em cascata, que foca em áreas com maior probabilidade de encontrar os *patterns*.

Enquanto isso, Karrach et al. [11] desenvolveu um método para identificar a distorção de perspectiva a partir da análise da direção das bordas horizontais e verticais e da maximização do desvio padrão das projeções horizontais e verticais dessas bordas.

O trabalhos de [12, 13] utilizam a detecção de bordas e a transformada de Hough para encontrar os lados do quadrilátero formado pelo QR Code. Não obstante, encontrar retas pela transformada de Hough demanda um elevado custo computacional, tornando-se inviável para aplicativos de leitura que desejam reconhecer o código bidimensional com rapidez em dispositivos celulares mais modestos.

Tendo isso em vista, Klimek e Vámossy [14] criaram um método para encontrar os lados de um QR Code por meio de uma variação da transformada de Hough, substituindo a parametrização tradicional que envolve equações trigonométricas por um sistema de coordenadas paralelas (*PClines*). Essa técnica otimiza o tempo de processamento em relação ao método de Hough original.

Suran [15] elaborou uma estratégia para determinar as distorções em QR Codes que se baseia em duas técnicas: no método de Harris para determinar pontos de quinas e no algoritmo envoltório convexo, que determina o menor polígono convexo formado pela união dos pontos de quinas. Após o formato do código QR ser encontrado, é aplicado uma transformação inversa de perspectiva que mapeia um quadrilátero arbitrário e, com o uso de mudanças de coordenadas, corrige a forma para um quadrado.

Neste artigo, com o intuito de remover a perspectiva em imagens de QR, desenvolveu-se um método que utiliza uma nova forma de encontrar os lados do polígono que evita, por exemplo, o aparecimento de retas que não fazem parte dos lados durante a determinação do quadrilátero, o que ocorre quando se usa a transformada de Hough e derivadas.

H. E. Gonçalves, Universidade Federal de Uberlândia, Uberlândia, MG, 38400-902, Brasil, e-mail: heitor.goncalves@ufu.br)

L. X. Medeiros, Universidade Federal de Uberlândia, Uberlândia, MG, 38400-902, Brasil, e-mail: lucianox@ufu.br

A. C. Mateus, Universidade Federal de Uberlândia, Uberlândia, MG, 38400-902, Brasil, e-mail: acmateus@ufu.br

Além disso, este artigo propõe uma técnica para remoção de distorções de modo que os módulos do QR Code apresentem, após a correção, o formato de um quadrado e com igual tamanho para todos, sem qualquer deformação.

Para identificar os métodos ao longo do artigo, a técnica para encontrar os vértices do QR Code será chamada de Retas por Degrais Consecutivos (RDC) e o artifício utilizado para remover a perspectiva será denominado Restauração dos Módulos por Substituição de Matriz (RMSM).

II. Fundamentos Teóricos

A. Conceitos Básicos de QR Codes

Os QR Codes são criados a partir de quadrados, conhecidos como *módulos*, que apresentam uma coloração escura ou clara, sendo que, comumente, utiliza-se as cores preta e branca. Esse tipo de código possui vários tamanhos que são chamados de *versões*, onde a Versão 1 possui dimensão de 21×21 módulos, a Versão 2 têm 25×25 módulos, seguida pela Versão 3 de 29×29, e assim por diante.

É importante ressaltar que um QR Code possui uma parte de sua imagem destinada aos códigos corretores de erros para que, mesmo se uma porção da superfície da imagem esteja danificada ou obstruída, a informação presente no QR Code possa ser decodificada. A técnica de correção de erros têm 4 níveis distintos possíveis, os quais são descritos na Tabela I, juntamente com a porcentagem de dados redundantes presentes na imagem.

TABELA I
Níveis de Correção de Erro.

Nível de correção de erro	Percentual de recuperação de dados
L	7%
M	15%
Q	25%
H	30%

A análise de regiões específicas do QR Code, os chamados *padrões funcionais*, permitem identificar as alterações de tamanho e inclinações dos módulos. Por esse motivo, a compreensão dessas regiões é imensamente importante para o entendimento do algoritmo que será apresentado na Seção III. Assim sendo, essas regiões estão representadas na Figura 1.

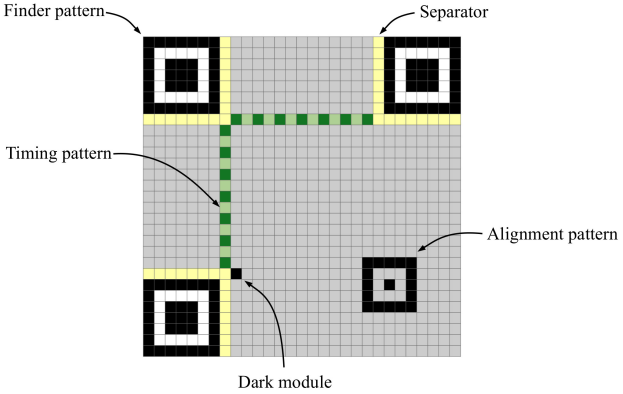


Fig. 1. Padrões Funcionais de um QR Code.

Na Figura 1 estão presentes os 5 tipos de padrões funcionais: *finder pattern*, *separator*, *timing pattern*, *dark module* e *alignment pattern*. Os módulos realçados em cinza são reservados tanto para armazenar dados a serem salvos no QR Code quanto os códigos corretores de erro. Os *separators* são destacados por quadrados amarelos e representam os módulos brancos que envolvem os *finder patterns* que, por sua vez, são os maiores quadrados presente na Figura 1, onde seus lados são formados por 7 módulos.

A partir da Versão 2, é possível encontrar os *alignment pattern*, cuja forma é semelhante aos *finders* de tamanho menor.

Outro padrão funcional importante para a execução do algoritmo proposto por esse trabalho são os *timing patterns* que são destacados na Figura 1 por quadrados em tons de verde claro e escuro e representam as faixas de módulos que se alternam entre as cores clara e escura, indo de um *separator* ao outro.

B. Equação da Reta

Considere que os pontos A e B pertencem ao plano cartesiano xy, como mostrado na Figura 2, onde (x_A, y_A) e (x_B, y_B) são as coordenadas dos pontos A e B, nessa ordem.

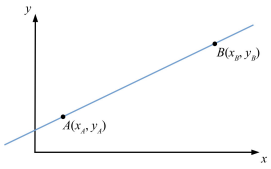


Fig. 2. Representação de uma reta no $\mathbb{R}^2$.

A equação da reta que passa por esses 2 pontos é dada por [16]:

$$y = m\,x + n \tag{1}$$

onde m e n são, respectivamente os coeficientes *angular* e *linear*. O valor de m pode ser calculado a partir das abcissas e ordenadas dos pontos A e B, por intermédio da Equação (2)

$$m = \frac{y_B - y_A}{x_B - x_A} \tag{2}$$

e quando o valor de m é conhecido, o coeficiente linear é determinado pela Equação (3)

$$n = y_A - m\,x_A \tag{3}$$

III. Algoritmo para encontrar os vértices do QR Code e remover a perspectiva

O algoritmo desenvolvido neste artigo possui 4 etapas: *Binarização da imagem*, *Determinação dos vértices*, *Remoção de Perspectiva* e *Correção da dimensão dos módulos*. Ressalta-se que as duas últimas etapas citadas constituem o método RMSM, enquanto na segunda etapa a técnica RDC é utilizada.

A. Binarização da Imagem

Em fotografias de QR Codes, é comum o aparecimento de diferentes tons de cinza devido a não uniformidade da iluminação ambiente. Portanto, o método proposto neste artigo necessita que a imagem seja binarizada, ou seja, convertida para uma escala de somente 2 níveis de cor, resultando assim em uma imagem a qual os seus píxeis podem assumir somente as cores preta e branca.

A binarização das fotos testadas nesse artigo ocorre por meio da função *im2bw*( ) e tal função está presente no pacote *image* 2.12.0 do GNU Octave, versão 5.2.0.

B. Determinação dos Vértices

Como os QR Codes possuem áreas quadradas, as deformações geométricas devido a perspectiva da fotografia resultam em um quadrilátero. Logo, esse polígono possui 4 vértices e 4 lados e pode-se considerar que cada lado é um segmento de reta. Para uma padronização do algoritmo proposto, nomeia-se os vértices de V_1, V_2, V_3 e V_4 e as retas de r_1, r_2, r_3 e r_4, conforme exemplificado pela Figura 3.

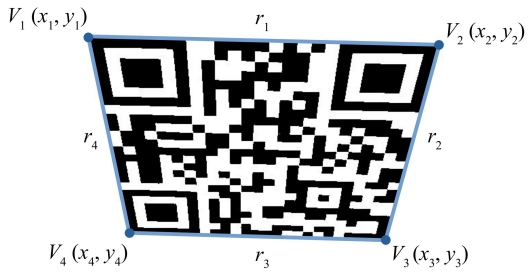


Fig. 3. Padronização dos vértices e retas de um QR Code.

Para localizar os vértices, é preciso, primeiramente, varrer a imagem por meio de 4 conjuntos de laços aninhados que a percorrem verticalmente e horizontalmente. Os primeiros píxeis de cor preta (valor numérico igual a zero), que são localizados pelos conjuntos de laços, têm suas posições armazenadas. O Algoritmo 1, que descreve esse processo, tem como saída os vetores V_1, V_2, V_3 e V_4, os quais possuem as coordenadas dos possíveis vértices do QR Code.

Após a execução do Algoritmo 1, o ponto central $P_{med}(x_{med}, y_{med})$ do QR Code é localizado com o auxílio da Equação (4).

$$x_{med} = \frac{1}{2}[\max(x_1, x_2, x_3, x_4) - \min(x_1, x_2, x_3, x_4)] + \min(x_1, x_2, x_3, x_4) \tag{4a}$$

$$y_{med} = \frac{1}{2}[\max(y_1, y_2, y_3, y_4) - \min(y_1, y_2, y_3, y_4)] + \min(y_1, y_2, y_3, y_4) \tag{4b}$$

onde max e min são, respectivamente, os operadores máximo e mínimo.

No Algoritmo 1 os pontos encontrados durante a varredura são armazenados nos vetores V_i, com ($i = 1, 2, 3$ e 4), onde a primeira e a segunda posição de cada um destes representa a coordenada em x e y, respectivamente.

Algoritmo 1: Localização dos vértices

```
1  var
2      i, j: inteiros
3      V1, V2, V3, V4: vetor[1..2] de inteiros
4      Img: matriz[1..L, 1..C] de inteiros
5  início
6      V1[1] ← −1;
7      V2[1] ← −1;
8      V3[1] ← −1;
9      V4[1] ← −1;
10     para i de 1 até L faça
11         para j de 1 até C faça
12             se Img[i, j] = 0 e V1[1] = −1 então
13                 V1[1] ← j;
14                 V1[2] ← i;
15             fim
16         fim
17     fim
18     para i de L até 1 passo −1 faça
19         para j de 1 até C faça
20             se Img[i, j] = 0 e V2[1] = −1 então
21                 V2[1] ← j;
22                 V2[2] ← i;
23             fim
24         fim
25     fim
26     para j de 1 até C faça
27         para i de 1 até L faça
28             se Img[i, j] = 0 e V3[1] = −1 então
29                 V3[1] ← j;
30                 V3[2] ← i;
31             fim
32         fim
33     fim
34     para j de C até 1 passo −1 faça
35         para i de 1 até L faça
36             se Img[i, j] = 0 e V4[1] = −1 então
37                 V4[1] ← j;
38                 V4[2] ← i;
39             fim
40         fim
41     fim
42 fim
```

Tendo como referência o ponto central, as posições coletadas são analisadas, de modo a determinar quais vértices foram obtidos pelo Algoritmo 1, além de rotulá-los de acordo com o padrão da Figura 3. Por exemplo, um ponto obtido que está acima e à esquerda de P_{med} na imagem é classifcado como V_1.

O V_3 pode ou não ter um pixel de cor preta, por isso esse procedimento não serve para sua identificação. Contudo, ressalta-se que, além do V_3, dependendo da distorção, o Algoritmo 1 pode também não conseguir coletar as posições de outros vértices. Logo, a varredura pode encontrar de 2 a 3 vértices.

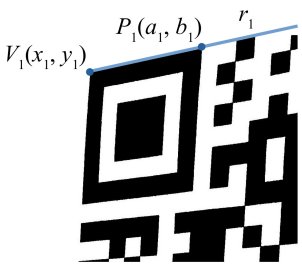


Fig. 4. Exemplo da reta r_1 que passa por $V_1(x_1, y_1)$ e $P_1(a_1, b_1)$.

Para detectar o V_3 e os outros vértices não localizados pelo Algoritmo 1, deve-se determinar as equações das retas r_1, r_2, r_3 e r_4, as quais podem ser escritas, de forma geral, como $r_i: y = m_i x + n_i$, onde m_i e n_i são, respectivamente, os coeficientes angular e linear da reta r_i, para $i = 1, 2, 3$ ou 4.

As retas são obtidas a partir das inclinações das bordas externas dos *finder patterns* onde o vértice é conhecido. Contudo, é fundamental verificar dentre os vértices localizados, quais podem ser utilizados para encontrar os coeficientes angulares das retas. Essa escolha tem como base o fato de que na varredura inicial, pelo menos um vértice da reta r_1 e da reta r_4 foram encontrados.

Além do vértice, a inclinação de um *finder pattern* necessita de um segundo ponto, o qual recebe a notação P_1. Este ponto é localizado percorrendo a borda do *finder pattern* até encontrar a fronteira com o *separator*, como mostrado na Figura 4.

O Algoritmo 2 exemplifica esse processo, além de encontrar a equação da reta r_1 a partir de V_1 (coordenada determinada pelo Algoritmo 1).

As determinações das retas r_2, r_3 e r_4, são feitas de forma análoga a r_1, isto é, obtém-se a inclinação da borda do *finder pattern* em relação à reta que se deseja calcular e os coeficientes angular e linear são calculados conforme as Equações 2 e 3, respectivamente.

Após concluir esse procedimento, é realizado uma nova varredura com direcionamento definido pelas equações das retas, permitindo encontrar os demais vértices que não foram localizados no escaneamento inicial, com excessão do V_3. Como V_3 é a intersecção entre as retas r_2 e r_3, as coordenadas desse ponto podem ser calculadas a partir da Equação (5).

$$x_3 = \frac{n_2 - n_3}{m_3 - m_2} \quad \text{e} \quad y_3 = \frac{n_2 m_3 - n_3 m_2}{m_3 - m_2} \tag{5}$$

C. Remoção de Perspectiva

Primeiramente, os 4 segmentos de retas (r_1, r_2, r_3 e r_4) são percorridos através de suas equações, por meio de estruturas de repetição, e as coordenadas de seus pontos são armazenadas em matriz. Importante salientar que esses segmentos são restringidos pelos vértices V_1, V_2, V_3 e V_4, padronizados pela Figura 3, os quais têm as suas abscissas e ordenadas computadas previamente.

Para realizar a correção vertical, isto é, para que as retas r_1 e r_3 fiquem paralelas ao eixo x, utiliza-se o Algoritmo 3. Esse algoritmo realiza uma varredura que passa por todas as retas que conectam o r_4 ao r_2 e salva os valores dos píxeis

Algoritmo 2: Cálculo da equação da reta de r_1

```
1  var
2      i, j, k, a1, b1, aux: inteiros
3      m1, n1: reais
4      V1: vetor[1..2] de inteiros
5      Img: matriz[1..L, 1..C] de inteiros
6  início
7      k ← 1;
8      b1 ← −1;
9      para j de V1[1] até C faça
10         para i de V1[2] até L faça
11             se Img[i, j] = 0 e b1 = −1 então
12                 aux[k] ← i;
13                 se k ≥ 2 e |aux[k] − aux[k − 1]| ≥ 2
                    então
14                     b1 ← j − 2;
15                 fim
16                 k ← k + 1;
17             fim
18         fim
19     fim
20     a1 ← −1;
21     para i de V1[2] até L faça
22         se Img[i, b1] = 0 e a1 = −1 então
23             a1 ← i;
24         fim
25     fim
26     m1 ← (a1 − V1[2])/(b1 − V1[1]);
27     n1 ← b1 − m1 ∗ a1;
28 fim
```

que foram percorridos em uma nova matriz, a qual agrupa o valor de cada reta em uma linha.

As Figuras 5(a) e 5(b) mostram, respectivamente, um QR Code antes e depois da correção vertical.

(a) (b) (c)

Fig. 5. (a) Exemplo de um QR Code em perspectiva e imagens resultantes das correções (b) vertical e (c) horizontal.

No Algoritmo 3 nota-se na Linha 26 o uso da função $arredonda(\ \ )$, que retorna o valor inteiro mais próximo da expressão numérica entre parênteses.

Após a correção vertical da imagem, mais uma vez o Algoritmo 1 é utilizado para encontrar as novas posições dos vértices. Logo em seguida, utiliza-se esses vértices para obter as retas r_2 e r_4. No caso das retas r_1 e r_3, novos coeficientes angulares são iguais a zero e coefientes lineares são as coodernadas em y de V_1 e V_4, respectivamente.

Em seguida, de forma semelhante ao que foi feito no Algoritmo 3, encontra-se as inclinações de retas que une

Algoritmo 3: Correção geométrica vertical

```
var
    i, j, C1, C2, L1, dif, tam: inteiros
    r, Δm, m3, m1, n4: reais
    V1: vetor[1..2] de inteiros
    m: vetor[1..tam] de reais
    Img: matriz[1..L, 1..C] de inteiros
    M: matriz[1..tam, 1..dif] de inteiros
início
    se (V1[1] < V4[1]) então
        C1 ← V1[1];
    senão
        C1 ← V4[1];
    fim
    se (V2[1] < V3[1]) então
        C2 ← V2[1];
    senão
        C2 ← V3[1];
    fim
    dif ← C2 − C1;
    tam ← V4[2] − V1[2];
    Δm ← m3 − m1;
    r ← Δm/(tam − 1);
    para i de 1 até tam faça
        m[i] = m1 + (i − 1) ∗ r;
        para j de 1 até dif faça
            L1 ← arredonda(m[i] ∗ (j + C1 − (i +
                 V1[2] − n4 − 1)/m4) + V1[2] + i);
            se L1 ≥ 1 e L1 ≤ L então
                M[i, j] ← Img[L1, j + C1];
            senão
                M[i, j] ← 255;
            fim
        fim
    fim
fim
```

todos os pontos de r_1 aos pontos de r_3. A posteriori, com o auxílio de um laço e as equações das retas obtidas, todas as coordenadas dos pontos pertencentes a cada uma das retas são salvas em uma nova matriz, que separa esses valores colocando uma coluna para cada reta. Esse processo permite a correção vertical da imagem. A Figura 5(c) apresenta um QR Code depois da correção horizontal.

D. Correção da Dimensão dos Módulos

As imagens resultantes após as 3 etapas anteriores do método proposto por esse artigo, são QR Codes com um formato retangular, similar ao mostrado na Figura 5(c) e dessa forma, é necessário tornar os módulos quadrados novamente. Porém, essa transformação não pode ser feita por uma simples alteração na escala horizontal ou vertical, isso porque os módulos apresentam tamanhos diferentes entre si. Assim, com o intuito de colocar todos os módulos com a mesma dimensão, é necessário calcular o comprimento vertical e horizontal de cada um destes. Para isso, as regiões destacadas na Figura 6 são percorridas.

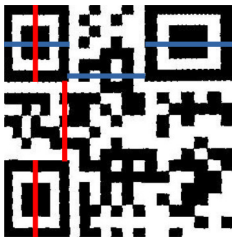


Fig. 6. Percurso utilizado para encontrar as dimensões dos módulos.

Na Figura 6 as linhas azuis reprezentam os caminhos percorridos para determinar a largura dos módulos, enquanto as linhas vermelhas são utilizadas para encontrar as alturas.

(a) (b)

Fig. 7. Exemplo de alternância de cores de uma mesma linha do (a) *finder pattern* e (b) *timing pattern*.

A Figura 7(a) apresenta 7 módulos consecutivos da linha que passa no centro de um *finder pattern*. Nota-se que é possível determinar o tamanho exato dos módulos 1, 2, 6 e 7 devido a transição de cores, porém como os módulos 3, 4, e 5 são todos de uma mesma cor, a largura média é igual ao comprimento total desses 3 módulos dividido por 3. Já a Figura 7(b) mostra o padrão de cores de um *timing pattern* e percebe-se nessa que é possível identificar o tamanho de todos os módulos, por causa das alternâncias de cores.

É possível determinar o tamanho do QR Code somando-se a quantidade de módulos na *timing patterns*, o que é feito utilizando um contador a cada alternância de cor, mais os 14 módulos de 2 *finder patterns* e os 2 módulos dos 2 *separator* que estão no início e fim de cada *timing*. Ao utilizar essa lógica para a Figura 6, tem-se 7 módulos nos *timing patterns* mais 14 nos *finder patterns*. Logo, esse código QR tem 21×21 módulos, correspondendo a Versão 1.

Em seguida, cria-se uma matriz com o número de píxeis igual ao tamanho do QR Code, onde cada módulo será representado por um único pixel. Destarte, a cor de cada pixel é determinado de acordo com um laço que percorre módulo por módulo da imagem após a remoção de perspectiva e armazena a cor do centro de cada um. Esse laço consegue ter a posição exata do centro dos módulos graças a determinação dos tamanhos destes. Foi escolhido utilizar os centros, pois nestas posições, existem uma menor possibilidade de encontrar ruidos, que são mais frequentes nas bordas.

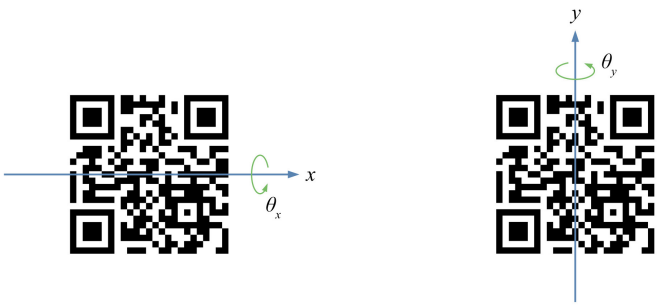
Posteriormente, cria-se uma região de cor branca em torno do QR Code, cuja a largura corresponde a 1 pixel e essa região é chamada de *Quiet Zone*. Por fim, altera-se a escala do QR Code de acordo com a dimensão da imagem original.

IV. Resultados

As imagens utilizadas nos testes foram criadas a partir do *software* GIMP 2.10 que possui uma ferramenta capaz de

alterar a perspectiva da imagem em relação a um eixo escolhido, especificando o valor do ângulo desejado. Outrossim, o algoritmo foi implementado e testado no GNU Octave 5.2.0, o qual estava sendo executado no sistema operacional Linux Mint 19.3, em um notebook com processador Intel Core i3-6006U e 4 GB de memória RAM.

Foram geradas imagens com perspectivas em relação à direção horizontal, considerando que o eixo x passa pelo centro do QR Code, como mostrado na Figura 8(a), e θ_x é o ângulo de inclinação em x, ou seja, o ângulo formado pelos planos que contém o QR Code e a câmera. Já as imagens com perspectiva em relação à direção vertical obedecem a mesma lógica, só que tendo como referência o eixo y, conforme a Figura 8(b), e nesse caso, o ângulo de inclinação é igual a θ_y. Todas as imagens utilizadas nos testes possuem 1200×1200 píxeis. Foram testados QR Codes com perspectivas no eixo x e/ou y da foto.



(a) (b)

Fig. 8. Perspectiva em relação aos eixos (a) x e (b) y.

Primeiramente, os testes foram realizados com 4 QR Codes distintos, onde cada um desses códigos possuia 8 imagens diferentes (8 valores de θ_x), de acordo com a Tabela II. Nessa tabela também são listadas as taxas de erro que são as porcentagens de píxeis das imagem resultantes do algoritmo proposto que foram classificados (ou preto ou branco) erroneamente, tendo como referência os QR Codes originais.

Nota-se na Tabela II que a perspectiva do QR Code 1 foi removida sem gerar erros para todos os ângulos. Já os outros QR Codes tiveram uma taxa de erro de 100% para $\theta_x = 75°$, o que é algo bem plausível, pois quanto maior for o ângulo de inclinação, mais os píxeis das linhas vizinhas começam a se sobrepor visualmente, impossibilitando a separação dos mesmos durante a remoção da perspectiva.

Curiosamente, o QR Code 4 teve taxas diferentes de zero para ângulos de 45°, 55° e 65° e isso é perfeitamente justificável, uma vez que cada módulo de um QR Code de Versão 10 (57×57 módulos) possui menos píxeis do que um código de Versão 1 (21×21 módulos), ou seja, mais pontos estão disponíveis para os cálculos que são realizados no método deste artigo para QR Codes de versões menores, claro que considerando uma mesma resolução.

A Figura 9 é uma forma de visualizar o funcionamento do método proposto, onde as Figuras 9(a) e 9(b) são, respectivamente, o QR Code com perspectiva em x e a imagem resultante do pós-processamento. A Figura 9(c) é a imagem diferença, onde os píxeis classificados incorretamente são destacados pela cor vermelha e como essa Figura não apresenta módulos vermelhos, isso indica que a perspectiva foi removida sem gerar erros, o que também pode ser constatado na Tabela II.

(a) (b) (c)

Fig. 9. Exemplo de perspectiva em relação a x, onde (a) QR Code 4 com $\theta_x = 55°$, (b) imagem resultante e (c) imagem diferença.

Os QR Codes 1, 2, 3 e 4 são testados novamente, mas agora com perspectivas em relação ao eixo vertical, onde θ_y também possui 8 valores para cada código e as taxas de erros são apresentadas na Tabela III.

Verifica-se na Tabela III que o algoritmo proposto não obteve erros para o QR Code 1, o que já era esperado por se tratar do código que tem a maior densidade de píxeis/módulo.

TABELA II
RESULTADO OBTIDOS EM x.

Nome da Imagem	Número de Módulos	Ângulo θ_x	Taxa de Erro
QR Code 1	21 × 21	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,00%
		55°	0,00%
		65°	0,00%
		75°	0,00%
QR Code 2	37 × 37	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,00%
		55°	0,00%
		65°	0,00%
		75°	100,00%
QR Code 3	41 × 41	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,00%
		55°	0,00%
		65°	10,28%
		75°	100,00%
QR Code 4	57 × 57	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,17%
		55°	0,03%
		65°	0,20%
		75°	100,00%

TABELA III
RESULTADO OBTIDOS EM y.

Nome da Imagem	Número de Módulos	Ângulo θ_y	Taxa de Erro
QR Code 1	21 × 21	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,00%
		55°	0,00%
		65°	0,00%
		75°	0,00%
QR Code 2	37 × 37	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,00%
		55°	0,00%
		65°	0,00%
		75°	100,00%
QR Code 3	41 × 41	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,00%
		55°	0,00%
		65°	0,00%
		75°	100,00%
QR Code 4	57 × 57	5°	0,00%
		15°	0,00%
		25°	0,00%
		35°	0,00%
		45°	0,00%
		55°	0,00%
		65°	1,09%
		75°	100,00%

Para $\theta_y = 75°$, a taxa de erro foi mais uma vez de 100% para os QR Codes 2, 3 e 4, e isso também já era aguardado, pois os píxeis de colunas contíguas se sobrepõe quando a inclinação aumenta e desse modo, os valores exatos das coordenadas x de cada pixel são perdidos.

A Figura 10(a) exibe um QR Code com $\theta_y = 65°$ e para esse exemplo, quando a técnica desenvolvida nesse trabalho é aplicada, são obtidos 1.09% dos módulos classificados erroneamente, os quais são destacados pela cor vermelha na Figura 10(c).

(a) (b) (c)

Fig. 10. Exemplo de perspectiva em relação a y, onde (a) QR Code 4 com $\theta_y = 65°$, (b) imagem resultante e (c) imagem diferença.

O algoritmo proposto neste artigo não se restringe a corrigir imagem com perspectiva somente em x ou em y, mas também é capaz de restaurar imagem que possua perspectiva em ambos os eixos simultaneamente, como mostra a Figura 11.

(a) (b)

Fig. 11. (a) Imagem com perspectiva, onde $\theta_x = 20°$ e $\theta_y = 30°$ e (b) imagem diferença.

Na Figura 11(a) observa-se uma imagem com $\theta_x = 20°$ e $\theta_y = 30°$. Após a aplicação do algoritmo, todos os módulos são posicionados corretamente, o que é visto na Figura 11(b).

Selecionou-se 3 aplicativos de leitura de QR Codes para dispositivos móveis, os quais podem ser obtidos na *Play Store* a partir dos links especificados no Apêndice A, para tentar decodificar as informações presentes nas mesmas fotografias com perspectiva que foram utilizadas nos testes anteriores. A porcentagem de imagens que tiveram suas informações extraídas são apresentadas na Tabela IV.

TABELA IV
LEITURA DE IMAGENS DEFORMADAS GEOMETRICAMENTE.

Aplicativo	Porcentagem de imagens que foram decodificadas
QR Code Reader	35,94%
QR & Barcode Scanner	25,00%
QR Scanner	25,00%

Verifica-se na Tabela IV que 2 aplicativos conseguiram porcentagens de decodificação de 25%, ou seja, a cada 4 imagens lidas, os aplicativos decodificaram somente 1. O aplicativo que obteve uma maior taxa foi o QR Code Reader com $35,94\%$, o que é próximo de 1/3 das imagens avaliadas.

Testando as imagens que tiveram as suas perspectivas removidas pelo algoritmo desenvolvido neste artigo e realizando a leitura das mesmas com os 3 aplicativos mencionados anteriormente, as taxas das imagens que foram decodificadas são exibidas na Tabela V.

TABELA V
LEITURA DE IMAGENS CORRIGIDAS PELO ALGORITMO.

Aplicativo	Porcentagem de imagens que foram decodificadas
QR Code Reader	85,84%
QR & Barcode Scanner	87,50%
QR Scanner	87,50%

Comparando as Tabela IV e V, nota-se um aumento significativo na taxa de decodificação das imagens pelos 3 softwares citados acima e demonstrando que o algoritmo proposto neste trabalho consegue melhorar bastante o número de imagens codificadas.

V. Conclusão

A leitura de um QR Code feita por meio de câmeras de celulares pode enfrentar problemas de decodificação causados por distorções geométricas, o que faz da remoção de perspectiva um tema pertinente. Pensando nisso, o algoritmo desenvolvido neste trabalho busca corrigir esse tipo de deformações e apresentou bons resultados para imagens de resoluções próximas ao HD (1.280×720 píxeis).

Verificou nos testes realizados que, quanto maior for o ângulo de inclinação, maior é a quantidade de módulos que são classificados erroneamente. Porém, para ângulos de aproximadamente 75° ou mais, o celular e a superfície onde está impresso o QR Code estão praticamente em planos perpendiculares, configurando uma situação que não aconteceria no dia a dia. Normalmente, os usuários que usam esse tipo de aplicativo posicionam os dispositivos móveis com inclinação menor do que 15° e para ângulos dessa magnitude, o método proposto consegue remover a perspectiva com eficácia.

Mesmo que as imagens resultantes da técnica de remoção de perspectiva deste artigo apresentem uma baixa taxa de módulos classificados errados, os programas de leitura de QR Codes ainda conseguem extrair as informações presentes nessas imagens devido aos códigos corretores de erro.

Uma forma de melhorar ainda mais a remoção de perspectiva pelo método proposto, seria aumentar a resolução das imagens testadas, assim o algoritmo teria mais pontos para realizar os cálculos necessários. Porém, a proposta desse artigo é trabalhar com imagem de dimensões intermediárias, mirando em dispositivos móveis com processamentos mais modestos e câmeras de resolução HD.

Apêndice A
Softwares Utilizados

1) GNU Octave: https://www.gnu.org/software/octave/
2) GIMP: https://www.gimp.org/
3) QR Code Reader: https://play.google.com/store/apps/detailsid=tw.mobileapp.qrcode.banner
4) QR & Barcode Scanner: https://play.google.com/store/apps/details?id=app.qrcode
5) QR Scanner: \https://play.google.com/store/apps/details?id=com.gamma.scan

Referências

[1] S. Morita and T. Eckschmidt, *QR Code: Comunicação e Engajamento*. CreateSpace Independent Publishing Platform, 2014.

[2] S. Tiwari, "An Introduction to QR Code Technology," in *2016 International Conference on Information Technology (ICIT)*, 2016, pp. 39–44.

[3] D. W. Incorporated, "History of QR Code," 2019. [Online]. Available: https://www.qrcode.com/en/history

[4] M. S. Hodage, M. S. Mangade, M. P. Touti, M. K. Yadav, and M. Kotkar, "Smart QR Code Based Application as Medicine Spotter for Visually Impaired," *International Journal of Advance Scientific Research and Engineering Trends*, vol. 3, no. 5, 2018.

[5] C.-y. Law and S. So, "QR codes in education," *Journal of Educational Technology Development and Exchange (JETDE)*, vol. 3, no. 1, p. 7, 2010.

[6] C.-T. Huang, Y.-H. Zhang, L.-C. Lin, W.-J. Wang, and S.-J. Wang, "Mutual authentications to parties with QR-code applications in mobile systems," *International Journal of Information Security*, vol. 16, no. 5, pp. 525–540, 2017.

[7] H. Zhang, C. Zhang, W. Yang, and C.-Y. Chen, "Localization and navigation using QR code for mobile robot in indoor environment," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2015, pp. 2501–2506.

[8] H. Tribak and Y. Zaz, "QR Code Recognition based on Principal Components Analysis Method," *(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 8, no. 4, 2017.

[9] T. Hicham and Y. Zaz, "QR code patterns localization based on Hu Invariant Moments," *Int. J. Adv. Comput. Sci. Appl*, vol. 8, no. 9, p. 162, 2017.

[10] L. Belussi and N. Hirata, "Fast QR Code Detection in Arbitrarily Acquired Images," pp. 281–288, 2011.

[11] L. Karrach, E. Pivarčiová, and P. Božek, "Identification of QR Code Perspective Distortion Based on Edge Directions and Edge Projections Analysis," *Journal of Imaging*, vol. 6, no. 7, p. 67, 2020.

[12] J. Zhou, Y. Liu, and A. Kumar, "Research on Distortion Correction of QR Code Images," 2012.

[13] J.-w. Wei, S.-g. Dai, and A. Mu, "Rectification And Localization of QR Code Image Based on Methmatical Morpholoy and Hough Transformation," *Computer and Information Technology*, vol. 6, 2010.

[14] G. Klimek and Z. Vamossy, "QR code detection using parallel lines," in *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE, 2013, pp. 477–481.

[15] K. Suran, "QR code image correction based on corner detection and convex hull algorithm," *Journal of multimedia*, vol. 8, no. 6, p. 662, 2013.

[16] P. Boulos and I. de Camargo E. Oliveira, *Geometria analítica: um tratamento vetorial*. Prentice Hall Brasil, 2005.

Heitor Eugênio Gonçalves Graduando em Engenharia Eletrônica e de Telecomunicações pela Universidade Federal de Uberlândia (UFU). Foi membro do Programa de Educação Tutorial (PET) Engenharia Elétrica durante o período de abril de 2019 a julho de 2020, onde atuou na organização dos eventos científicos XVII Conferência de Estudos em Engenharia Elétrica e XVI Jornada da Engenharia Elétrica, de minicursos e de programas de extensão, como o Engenharia no Ensino Médio.

Luciano Xavier Medeiros Possui graduação em Engenharia Elétrica com ênfase em Eletrônica pela Universidade Federal de Uberlândia (2003), mestrado em Engenharia Elétrica pela Universidade Federal de Uberlândia (2006) e doutorado em Engenharia Elétrica pela Universidade Federal de Uberlândia (2012). Durante o período de 2010 a 2017, fez parte do quadro de professores efetivos da Universidade Federal do Triângulo Mineiro (UFTM), onde foi membro do Departamento de Engenharia Elétrica dessa instituição, lecionando diversas disciplinas, como Eletromagnetismo, Linhas de Transmissão e Radiação, Eletrônica Digital 1 e 2, Processamento Digital de Imagens, dentre outras. Atualmente é professor adjunto da Universidade Federal de Uberlândia (UFU), atuando principalmente na área de Engenharia de Telecomunicações.

Alexandre Coutinho Mateus Possui graduação em engenharia elétrica pela Universidade Federal de Uberlândia (1996), mestrado em engenharia elétrica pela Universidade Federal de Uberlândia (1998) onde atuou na área de Automação e Controle desenvolvendo um software de controle de nível utilizando lógica Fuzzy em uma rede fieldbus; concluiu doutorado em engenharia elétrica pela Universidade Federal de Uberlândia (2007), onde desenvolveu estudos na área de Telecomunicação trabalhando com propagação na baixa ionosfera, iniciou o curso de pós-doutorado na engenharia mecânica pela Universidade Federal de Uberlândia, (2009)participando de um projeto de pesquisa para construção de uma máquina de ensaio de fadiga dedicada a sistemas biomecânicos, no núcleo de projetos e sistemas mecânicos ; atualmente professor adjunto no departamento de engenharia Elétrica da Fundação Universidade Federal de Uberlândia.