Transformações Geométricas

Introdução ao Processamento de Imagem Digital

Percy Maldonado Quispe
203968

1. Introdução

As transformações geométricas são operações que permitem o mapeamento entre as posições espaciais x, y de uma imagem inicial f para uma imagem final g que foi modificada por uma matriz ou transformada, mas as transformações geométricas não são apenas mapeamento 3.2, mas também tem a parte onde você deve realizar uma interpolação 4 para encontrar o nível de cinza que melhor corresponde às posições mapeadas no passo anterior.

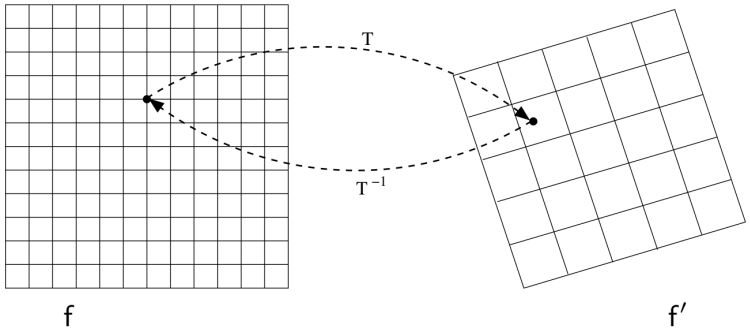


Figura 1: Transformação geométrica

Para mapear os pontos de uma imagem inicial para uma imagem final, diferentes transformações podem ser aplicadas, dentre as mais conhecidas temos as seguintes, além de métodos de interpolação:

Transformações

- Escala.
- Translação.
- Rotação.
- Reflexão.
- Cisalhamento.

Métodos de interpolação

- Nearest Neighbor.
- Bilinear.
- Bicubic.
- Lagrange.

Para o desenvolvimento deste trabalho, foi realizada a implementação de todas essas transformadas e métodos de interpolação.

2. Características do Programa

O programa foi feito em linguagem Python, com bibliotecas externas NumPy e OpenCV. Os argumentos são passados em tempo de execução via *argv*, que permite modificar várias condições em tempo de execução.

Os argumentos do programa podem ser:

Figura 2: Numpy, Python e OpenCV

- *-a/–angle*: Ângulo de rotação em graus (sentido anti-horário).
- *-s/–scale*: Vetor de escala S_x, S_y para cada eixo.
- *-t/–translation*: Vetor de translação T_x, T_y, valores a serem movidos em cada eixo.
- *-r/–reflection*: Vetor de reflexão R_x, R_y para cada eixo, valores permitidos (-1,1), onde -1 indica o eixo onde a reflexão será aplicada.
- *-k/–shear*: Vetor de cisalhamento K_x, K_y para cada eixo, pode assumir valores entre (0,1).
- *-d/–dimension*: Dimensão na qual a imagem de saída será mapeada N x M.
- *-m/–method*: Método de interpolação, os valores permitidos são: *(nearest, bilinear, bicubic y lagrange)*.
- *-i/–image*: Path da imagem de entrada.
- *-o/–output*: Path onde a imagem transformada será salva de acordo com os parâmetros indicados.

3. Transformações Geométricas

É uma transformação 1 que se aplica às coordenadas de um pixel na imagem, modificando sua localização.

$$x = T(x') \tag{1}$$

3.1. Projeção de perspectiva

A projeção em perspectiva é um problema de mapeamento de posições que estão contidas entre 4 pontos para outra que também está contida entre 4 pontos.

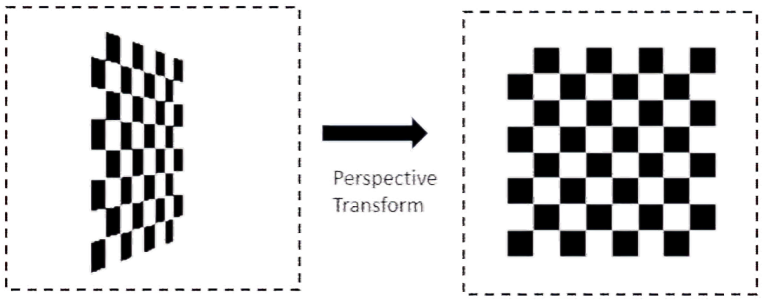


Figura 3: Transformação de perspectiva

Para resolver este problema de perspectiva, devemos primeiro encontrar a matriz *perspective* que nos permite mapear os pontos seguindo a equação 2, por isso precisamos de 4 pontos para poder resolver este sistema de equações de 8 variáveis (a, b, c, d, e, f, i, j).

$$X' = \frac{aX + bY + c}{iX - jY + 1}$$
$$Y' = \frac{dX + eY + f}{iX - jY + 1} \tag{2}$$

Uma vez que tenhamos a matriz T, podemos mapear as posições para a imagem final, para este procedimento temos duas opções:

$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ i & j & 1 \end{bmatrix} \tag{3}$$

Características dos modos de aplicação da transformação:

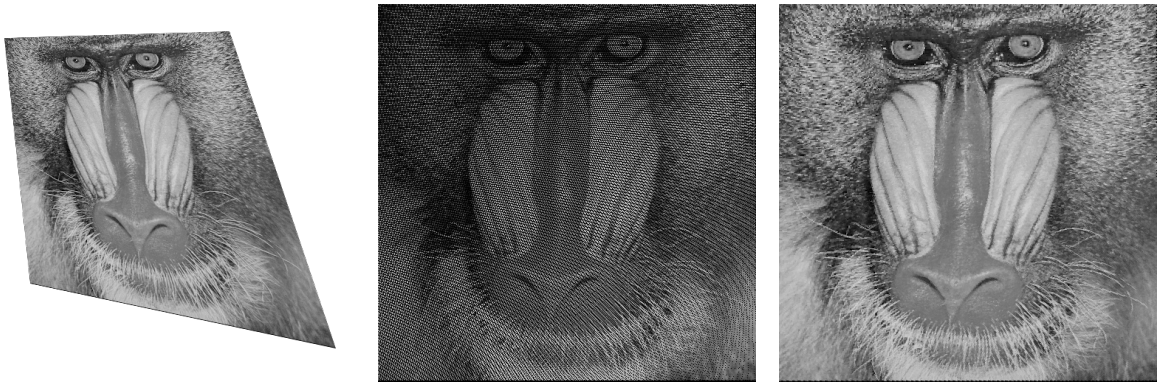
Transformação Direta: $P' = TP$

1. A localização não é exatamente um inteiro.
2. Valores podem não ser gerados em determinados pixeis e podem aparecer artefatos.

Transformação Reversa: $P = T^{-1}P'$

- Garante que toda a imagem este completa.
- Requer técnicas de interpolação.

Os resultados nestes dois modos de aplicação da transformada são bem diferentes, como pode ser visto na imagem 4. No modo direto 4b, é possível ver buracos que não podem ser mapeados da imagem inicial para a imagem final, que na transformação inversa 4c não aparece porque realiza um travessia na imagem final e não na entrada como 4b faz.



(a) Original (b) $P' = TP$ (c) $P = T^{-1}P'$

Figura 4: 4a) imagem original, 4b), 4c) resultado da aplicação da transformação direta e inversa (*nearest neighbor interpolation*) respectivamente

3.2. Transformações

Quando se trata de transformações, temos um grande número de transformações já definidas para diferentes tipos de aplicações ou necessidades.

3.2.1. Escala

Esta transformação permite o redimensionamento de uma imagem de acordo com o fator de escala escolhido, S_x, S_y no eixo x ou no eixo y respectivamente.

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

3.2.2. Translação

No caso de tradução, também pode ser aplicado a ambos os eixos T_x, T_y

$$T = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

3.2.3. Rotação

Para a transformação de rotação é necessário especificar o ângulo que θ vai girar, neste trabalho o ponto de rotação é a coordenada (0,0).

$$S = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

3.2.4. Reflexão

Para a reflexão, pode ser especificado pelos parâmetros R_x para uma reflexão no eixo x e R_y para uma reflexão no eixo y, deve-se notar que os valores permitidos são -1 para uma reflexão e 1 para uma não reflexão.

$$R = \begin{bmatrix} R_x & 0 & 0 \\ 0 & R_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

3.2.5. Cisalhamento

Em relação ao cisalhamento, também são necessários os fatores de cisalhamento para ambos os eixos x, y.

$$K = \begin{bmatrix} 1 & K_x & 0 \\ K_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

4. Métodos de interpolação

A equação 9 é a realizada no passo anterior, que é aplicar a transformação do espaço de saída (imagem transformada) à imagem original, onde o valor de x, y são decimais e nós precisa obter a intensidade nesse ponto, dependendo de algum método de interpolação.

$$T^{-1}(x', y') = (x, y) \tag{9}$$

4.1. Nearest Neighbor

Para interpolação do vizinho mais próximo, simplesmente arredonde o resultado (x, y) que veio de (x', y') aplicando a transformação inversa.

$$g(x', y') = f(round(x), round(y)) \tag{10}$$

4.2. Bilinear

Este método de interpolação pega os 4 pontos mais próximos e pondera os níveis de cinza de acordo com a distância entre o ponto (x, y).

$$\begin{aligned} g(x', y') =& (1 - dx)(1 - dy)f(x, y) + dx(1 - dy)f(x + 1, y) + \\ & (1 - dx)dyf(x, y + 1) + dxdyf(x + 1, y + 1) \end{aligned} \tag{11}$$

4.3. Bicubic

A interpolação bicúbica usa uma vizinhança de 4×4 pontos ao redor do ponto em questão para calcular seu valor de intensidade. Para o cálculo dessas intensidades, foi utilizada a implementação da matriz, conforme apresentado por [Koljonen et al., 2019]

$$
\begin{aligned}
f(x,y) &= \sum_{i=0}^{2}\sum_{j=0}^{2} a_{ij}x_i y_j \\
f(x,y) &= \begin{bmatrix} u(x_3) & u(x_2) & u(x_1) & u(x_0) \end{bmatrix} \begin{bmatrix} a_{3,3} & a_{3,2} & a_{3,1} & a_{3,0} \\ a_{2,3} & a_{2,2} & a_{2,1} & a_{2,0} \\ a_{1,3} & a_{1,2} & a_{1,1} & a_{1,0} \\ a_{0,3} & a_{0,2} & a_{0,1} & a_{0,0} \end{bmatrix} \begin{bmatrix} u(y_3) \\ u(y_2) \\ u(y_1) \\ u(y_0) \end{bmatrix} \\
u(t) &= \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & \text{for } |x| \leq 1, \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & \text{for } |x| < 2, \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{12}
$$

Onde a geralmente assume o valor de -0,5 ou -0,75.

4.4. Lagrange

O método de Lagrange foi claramente implementado seguindo a seguinte fórmula proposta nas aulas.

$$
\begin{aligned}
g(x',y') =& \frac{-dy(dy-1)(dy-2)L(1)}{6} + \frac{(dy+1)(dy-1)(dy-2)L(2)}{2} + \\
& \frac{-dy(dy+1)(dy-2)L(3)}{2} + \frac{dy(dy+1)(dy-1)L(4)}{6} \\
L(n) =& \frac{-dx(dx-1)(dx-2)f(x-1,y+n-2)}{6} + \frac{(dx+1)(dx-1)(dx-2)f(x,y+n-2)}{2} + \\
& \frac{-dx(dx+1)(dx-2)f(x+1,y+n-2)}{2} + \frac{dx(dx+1)(dx-1)f(x+2,y+n-2)}{6}
\end{aligned}
\tag{13}
$$

No entanto, ao olhar para os resultados, apresentou artefatos notórios e ainda mais do que a interpolação *Nearest Neighbor*, por isso outra implementação foi feita de acordo com [Bozorgmanesh et al., 2009].

5. Implementação

Na implementação, foi feito de forma inversa, para poder mapear toda a imagem de saída e obter o valor correspondente da imagem inicial.

1. Transformação

Listing 1: Transform

```
# apply transform
def transform(img, M, dsize, inter):
    # indices [i, j, 1]
    iY, iX = np.indices(dimensions=dsize)
    indexOutput = np.stack(
        (iX.ravel(), iY.ravel(), np.ones(iY.size))).astype(int)

    # inverse matrix
    IM = np.linalg.inv(M)

    # dot product to move position
    indexInput = IM.dot(indexOutput)
    indexInput /= indexInput[2, :]

    # interpolation method
```

```
    if inter == 'nearest':
        out = inter_nearest(img, indexInput)
    elif inter == 'bilinear':
        out = inter_bilinear(img, indexInput)
    elif inter == 'bicubic':
        out = inter_bicubic(img, indexInput)
    elif inter == 'lagrange':
        out = inter_lagrange(img, indexInput)
    else:
        return np.zeros(dsize)

    return out.reshape(dsize)
```

Como pode ser visto no código python apresentado acima, a transformação das posições é realizada realizando uma única multiplicação de matrizes, isso gera uma vantagem em termos de tempo de execução. Da mesma forma, é possível observar que a função 1 suporta um grande número de transformações, portanto foi possível incluir operações como tradução, reflexão, cisalhamento.

5.1. Execução

Para executar os métodos descritos acima, podemos fazê-lo a partir de um terminal que tenha os seguintes pacotes instalados:

- *opencv-python*
- *matplotlib*
- *numpy*

Aqui está um exemplo de como aplicar uma transformação em uma imagem de entrada e como podemos usar os parâmetros suportados que mencionamos na seção 2.

6. Resultados e Comparações

Os resultados foram obtidos para 6 imagens diferentes (n), com 5 transformações diferentes 3.2 (m), 4 métodos de interpolação diferentes 4 (p) também foram aplicados. O número total de imagens de saída é: $n * m * p$, onde n é o número de imagens, m o número de transformações e p o número de métodos de interpolação, dando um total de 120 imagens de saída. As imagens de entrada estão disponíveis no seguinte link https://www.ic.unicamp.br/~helio/imagens_png/.

6.1. Comparação

Nesta seção faremos uma comparação entre os métodos de interpolação:

6.1.1. Comp. entre métodos de interpolação

- **Nearest Neighbor**: Este método é simples e fácil de implementar e nenhum dado artificial é introduzido na imagem de saída. No entanto, a imagem de saída contém artefatos que podem ter efeitos negativos dependendo do uso dessas imagens resultantes. Além disso, este método faz com que as linhas diagonais presentes na imagem agora mostrem uma forma de "escadaria", como pode ser visto na imagem 8a.
- **Bilinear**: Isso geralmente é bom para redimensionar uma imagem ou aplicar uma transformação, mas também causa suavização de detalhes que pode ser irregular. Isso resulta em imagens muito mais suaves do que o vizinho mais próximo.
- **Bicubic**: Este método vai um passo além do *Bilinear* ao considerar a vizinhança 4x4 mais próxima do pixel conhecido, para um total de 16 pixels. *Bicubic* produz imagens mais nítidas do que os dois métodos anteriores.
- **Lagrange**: Este método apresenta uma suavização mais suave que o *Bicubic*, porém demora um pouco mais para ser executado.

6.1.2. Comp. entre o tempo de execução

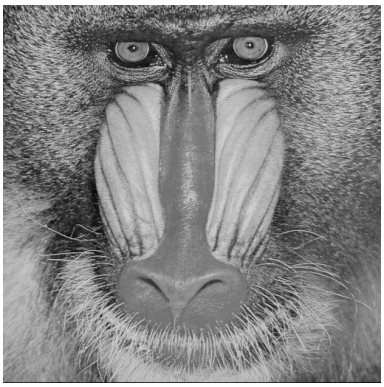
Seguindo nossa implementação dos métodos de interpolação, temos os seguintes resultados mostrados na tabela 1, onde você pode ver que o método de interpolação *Lagrange* leva mais tempo , isso porque nós fizemos não conseguimos vetorizar este método, porém os métodos *Nearest Neighbor* e *Bilinear* foram otimizados usando vetorização, apenas realizando arredondamentos e operações como somas/subtrações entre todos os índices respectivamente. No caso do método *Bicubic*, a implementação foi realizada de forma matricial para cada ponto x', y' que está dentro da imagem inicial, porém ainda há um atraso na execução, esses métodos ainda pode ser otimizado, como é o caso do *Bicubic*, que no *OpenCV* faz isso em apenas alguns segundos.

	Nearest	Bilinear	Bicubic	Lagrange
Escala (0.75,0.75)	0.0229	0.04162	6.394	8.809
Translação (50,50)	0.02225	0.04384	9.84	12.26
Rotación 15^o	0.02318	0.04134	7.831	12.11
Reflexión (1.0, -1.0)	0.02968	0.04262	0.02078	0.01918
Shear (0.25, 0.25)	0.0229	0.04116	8.805	10.86

Cuadro 1: Tempo de execução da imagem *baboon* de 512x512.

6.1.3. Resultados

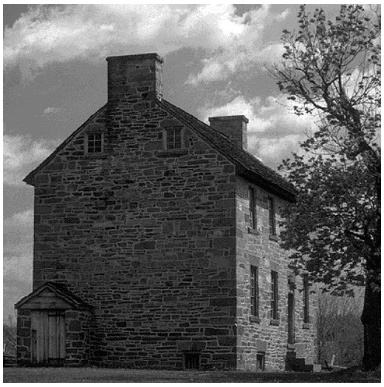
Nesta seção podemos ver como os métodos de interpolação se comportam para as imagens de entrada 5.



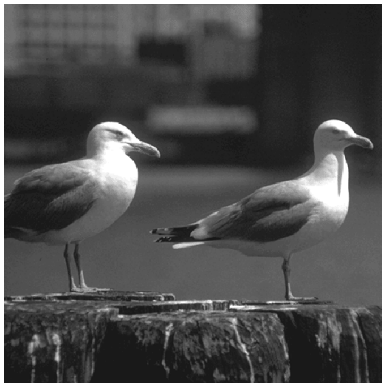
(a) Baboon

(b) Butterfly

(c) City



(d) House



(e) Seagull

(f) Trip

Figura 5: Imagens de entrada

Transformação

- Rotação da imagem *Baboon* 7.

- Rotação da imagem *House* 8.
- Transformações para a imagem *butterfly*, aplicando interpolação bicúbica 6:
  - Rotação de 15^o.
  - Escala (0.75,0.75).
  - Translação (50,50).
  - Cisalhamento (0.25,0.25).

(a) Rotação (b) Escala (c) Translação (d) Cisalhamento

Figura 6: Transformações aplicadas à imagem *Butterfly*

Devemos ressaltar que temos o parâmetro *-d/–dimension*, que nos permite obter a imagem em uma dimensão maior, na qual as transformações como *Cisalhamento* podem ser observadas em sua totalidade.

7. Conclusões

Para encontrar a matriz *perspective*, podemos calcular assim que tivermos os 4 pontos que a contém, com isso podemos resolver esse sistema de equações, também vimos que a transformação direta deixa lacunas e não completa a saída imagem, por isso é realizada a transformada inversa, que requer um método de interpolação.

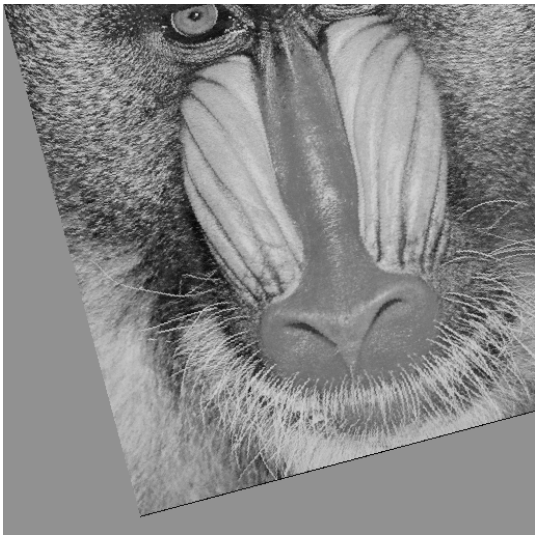
Em relação à transformada inversa e aos métodos de interpolação, podemos observar diferentes comportamentos na imagem resultante. Com base nos experimentos realizados, podemos concluir o seguinte:

- A transformada inversa permite mapear todos os pontos da imagem de saída.
- O método *Nearest Neighbor* é simples e rápido, pois apenas uma operação de arredondamento é realizada, este método deixa artefatos perceptíveis nas imagens de saída.
- *Bilinear* pega os 4 vizinhos mais próximos e suaviza a imagem de saída, em vez disso *Bicubic* pega os 16 vizinhos mais próximos e melhora a suavização ainda melhor em comparação com *Bilinear*, isso devido à sua natureza de graus de distância em relação ao ponto x, y.

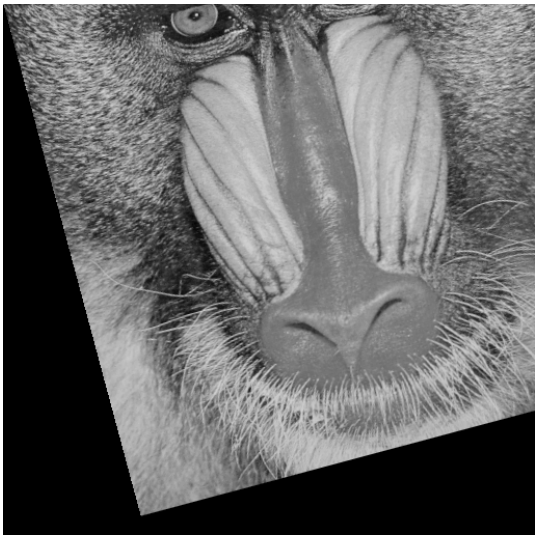
Além dos pontos mencionados, qualquer tipo de transformação pode ser aplicada em coordenadas homogêneas, conforme descrito na seção 3.2. A implementação faz uso de recursos *Numpy* como *vectorization* e *broadcasting*.

Referencias

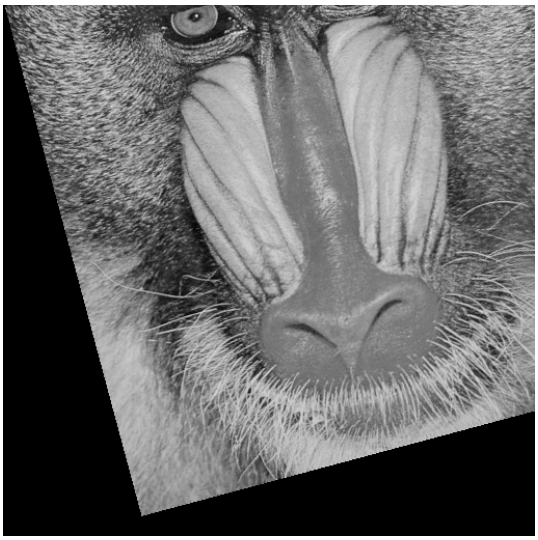
[Bozorgmanesh et al., 2009] Bozorgmanesh, A., Otadi, M., SAFE, K. A., Zabihi, F., and BARKHORDARI, A. M. (2009). Lagrange two-dimensional interpolation method for modeling nanoparticle formation during ress process.



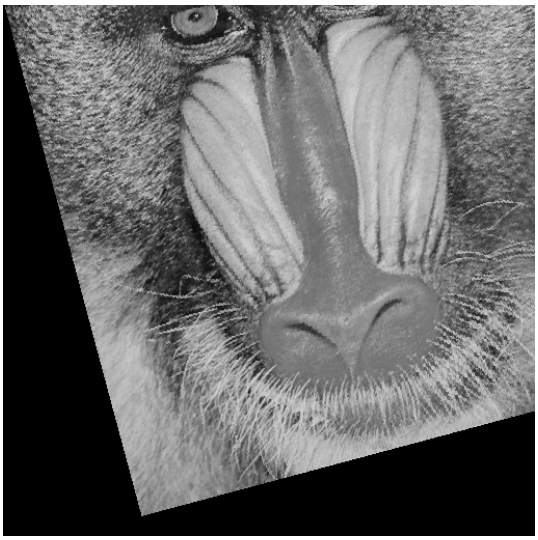
(a) Nearest Neighbor



(b) Bilinear



(c) Bicubic



(d) Lagrange

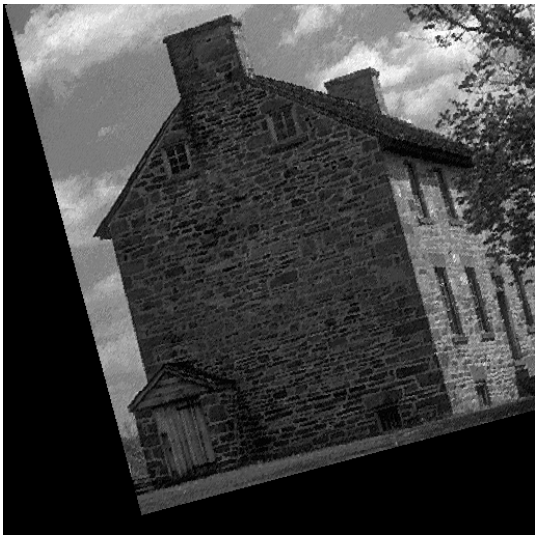
Figura 7: Resultado da aplicação dos diferentes métodos de interpolação para a imagem *baboon* ao girar 15^o

[Koljonen et al., 2019] Koljonen, J., Bochko, V. A., Lauronen, S. J., and Alander, J. T. (2019). Fast fixed-point bicubic interpolation algorithm on fpga. In *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pages 1–7.

(a) Nearest Neighbor

(b) Bilinear

(c) Bicubic



(d) Lagrange

Figura 8: Resultado da aplicação dos diferentes métodos de interpolação para a imagem *house* ao girar 15^o