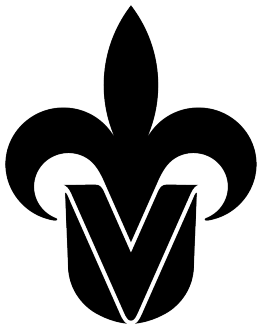

Universidad Veracruzana

Maestría en Inteligencia Artificial
Visión por Computadora

Tarea 7
Transformaciones afines en imágenes

Ulises Jiménez Guerrero

21 de abril de 2025

Reporte tarea 7

1. Objetivos

- Escribir un programa que implemente transformaciones afines. Las entradas del programa serán las imágenes 2D y los parámetros de la transformación (T_x, T_y, S_{xy}, Rot).
- Mejorar la imagen resultante con el programa de interpolación bilineal realizado anteriormente (Tarea 2).

2. Metodología

2.1. Materiales utilizados

Se utilizó el lenguaje de programación de Matlab, versión académica R2024b. No se requirió el uso de toolboxes o programas externos, restringiéndose al uso de funciones base de Matlab. Se utilizaron imágenes de autoría propia para comprobar el desempeño del código realizado, mostradas en 1 en su tamaño original. A su vez, se reutilizó parte del código desarrollado en la tarea 2, traduciéndolo a Matlab, para su uso en la interpolación bilineal.

Como referencia, se tiene al libro base del curso [2] para el estudio teórico de las transformaciones afines. Como complemente, se utilizó [1] como una explicación de la implementación de estas transformaciones en Matlab.

2.2. Base teórica

Las *transformaciones afines* son parte las transformaciones geométricas de imágenes, con la particularidad de que preservan puntos, líneas y planos. Este tipo de transformaciones se compone de diferentes operaciones sobre la imagen: *traslación, escalado, rotación y cizallamiento (shearing).* Se componen de dos partes, la transformación de coordenadas y la reasignación de intensidad o color.

La transformación de coordenadas se puede expresar como un producto matricial, simplificando su cálculo. Sin embargo, presenta problemas a la hora de lidiar con la traslación, dado que esta requiere una suma de vectores.

Para solucionar este inconveniente, se utilizan *coordenadas homogéneas*, agregando una dimensión extra a las coordenadas originales. Dado que en nuestro caso se trabaja con coordenadas $(x, y)^T$, se expanden a tres dimensiones, obteniendo coordenadas $(x, y, 1)^T$. De esta manera, todas las transformaciones afines se pueden expresar utilizando una matriz 3×3 de la forma

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{1}$$

donde $(x, y, 1)$ indica la coordenada homogénea en la imagen original y $(x', y', 1)$ la coordenada homogénea en la imagen resultante después de aplicar la transformación.

(a) Primera imagen a transformar, mostrando las letras de la ciudad de Orizaba. Nótese que es bastante más ancha que alta. Tamaño original: 154×500.

(b) Seguna imagen a transformar, mostrando a un cocker spaniel sonriendo. Nótese que es casi cuadrada. Tamaño original: 325×308.

Figura 1: Imágenes de autoría propia utilizadas para comprobar el programa desarrollado.

Cada operación tiene su matriz correspondiente, y se pueden componer diferentes operaciones realizando el producto matricial correspondiente. Es necesario destacar que el orden de operaciones es importante, y cambia de manera significativa el resultado final.

Para la reasignación de color, usualmente se sigue un proceso de interpolación similar al seguido en la tarea 2 de este curso, trabajando sobre el píxel correspondiente en la imagen original.

Las transformaciones afines se pueden realizar de dos maneras: mapeo hacia adelante y mapeo hacia atrás. En el mapeo hacia adelante, se itera sobre cada píxel en la imagen original, calculando su posición correspondiente en la imagen resultante mediante aplicación directa de la ecuación 1, y asignando la intensidad correspondiente. A pesar de ser una técnica directa y sencilla, presenta problemas como el hecho de que diferentes píxeles en la imagen original pueden llevar al mismo píxel en la imagen resultante, o que existan píxeles sin intensidad asignada.

El mapeo hacia atrás lidia con estos problemas iterando sobre la imagen resultante, y encontrando el píxel correpondiente en la imagen original resolviendo $(x, y) = \mathbf{A}^{-1}(x', y')$. Después de esto, la intensidad se obtiene mediante una interpolación. Esta suele ser la técnica preferida para implementaciones comerciales, y la utilizada en este proyecto.

A continuación se muestran las matrices correspondientes a cada una de las transformaciones afines utilizadas en este proyecto, junto a los parámetros correspondientes. Nótese que no se trabajo con *shearing*, limitándonos a traslación, escalado y rotación.

Transformación	Matriz de transformación	Parámetros
Traslación	$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$	T_x indica el desplazo en el eje X. T_y indica el desplazo en el eje Y.
Escalado	$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	S_x indica el factor de escala en el eje X. S_y indica el factor de escala en el eje Y.
Rotación	$\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	θ indica el ángulo de rotación.

2.3. Implementación

Se programó la función `affineTransform(img, tx, ty, scale, theta)`, que toma una imagen en su forma matricial, y aplica las transformaciones afines en el orden indicado en sus argumentos. Esto es, primero realiza una traslación, posteriormente reescala la imagen y finalmente la rota. Como se mencionó en la sección anterior, se utilizó la técnica de mapeo hacia atrás, para evitar los problemas asociados con el mapeo hacia adelante. Sin embargo, esto implica que se necesita elegir el tamaño de la imagen resultante al inicio de la transformación. Se decidió que esta tenga el tamaño de la imagen original, lo que implica que puede salir del rango considerado. El escalado se restringe a tomar el mismo valor tanto en el eje X como en el eje Y.

Para el proceso de transformación de la imagen, se empieza generando las matrices correspondientes a cada operación con los parámetros indicados en las entradas de la función, y la matriz de transformación final se obtiene mediante el producto matricial de estas. Posteriormente, se procede a iterar sobre cada uno de los píxeles de la imagen original, computando su coordenada correspondiente en la imagen original mediante la instrucción
`og_pos = trans_matrix\new_pos`. Se evita el uso de la matriz inversa dado que Matlab indica que su implementación es poco exacta e inestable.

Teniendo las coordenadas (x, y) del píxel correspondiente en la imagen original, se realiza un interpolación para obtener la intensidad que se le debe asignar en la imagen resultante. No se puede utilizar de manera directa la función realizada en la tarea 2, dado que esta toma como entrada imágenes enteras, y en este caso se interpola sobre únicamente un píxel. Por lo tanto, se tomó la parte correspondiente del código y se adaptó. Para esto, primero se requirió traducir la implementación de la interpolación de Python a Matlab.

Obteniendo la intensidad, se le asigna al píxel correspondiente en la imagen resultante y se itera al siguiente punto. Para los casos cuando el píxel correspondiente en la imagen original no se encuentra dentro de sus límites, se salta a la siguiente iteración. Dado que la imagen resultante se genera inicialmente como una matriz de ceros, esto significa que a dicho pixel se le asignará la tonalidad de negro.

3. Resultados

En las figuras 2, 3, 4, 5 y 6 se comparaciones entre las imágenes obtenidas después de aplicar la transformación y las imágenes originales, indicando los parámetros utilizados en cada caso y comentarios pertinentes. Algo a destacar es la diferencia entre los ejes de coordenada utilizados en imágenes y aquellos utilizados en otras áreas: en imágenes, el origen se encuentra en la esquina superior izquierda, y el eje X recorre la imagen de arriba a abajo, mientras que el eje Y recorre los píxeles de izquierda a derecha. Por lo tanto, a la hora de realizar una rotación, esta se hará tomando como pivote a la esquina superior izquierda de la imagen.

Figura 2: Imagen 1 trasladada en el eje X e Y. Parte de la imagen sale de los límites de la imagen, por lo que se muestra como píxeles negros. $T_x = 20,\ T_y = 20,\ S_{xy} = 1,\ \theta = 0$.

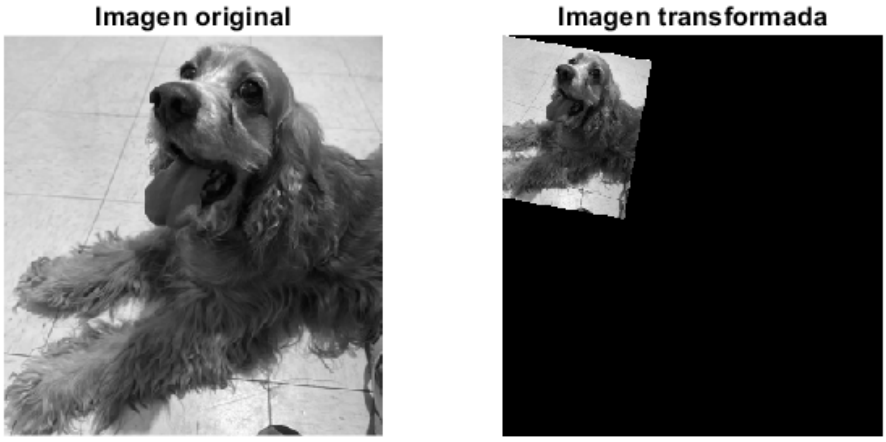

Figura 3: Imagen 2 escalada a un tamaño menor y rotada ligeramente. Dado que el pivote se encuentra en la esquina superior izquierda, se empieza a perder parte de la imagen. Si se aumenta el ángulo de rotación sin realizar traslaciones, se perderá completamente. Dado el grado de compresión, se pierde bastante calidad de la imagen. $T_x = 0,\ T_y = 0,\ S_{xy} = 0.4,\ \theta = 10$.

Figura 4: Imagen 2 escalada a un tamaño mayor. Mediante el uso correcto de traslaciones, se puede observar con mayor detalle ciertas zonas de la imagen, en este caso, la cara del perro. Nuevamente se pierde calidad a pesar del uso de la interpolación bilineal. $T_x = -30,\ T_y = -200,\ S_{xy} = 2,\ \theta = 0$.

Figura 5: Imagen 1 trasladada, reescalada y rotada. Se utilizaron números no enteros para mostras la capacidad del programa desarrollado para lidiar con este caso. Dada la forma de la imagen, una rotación muy leve la hace perder información, sin que exista una traslación que la ajuste de manera adecuada. $T_x = 5.3,\ T_y = 215.82,\ S_{xy} = 0.4,\ \theta = 45.$

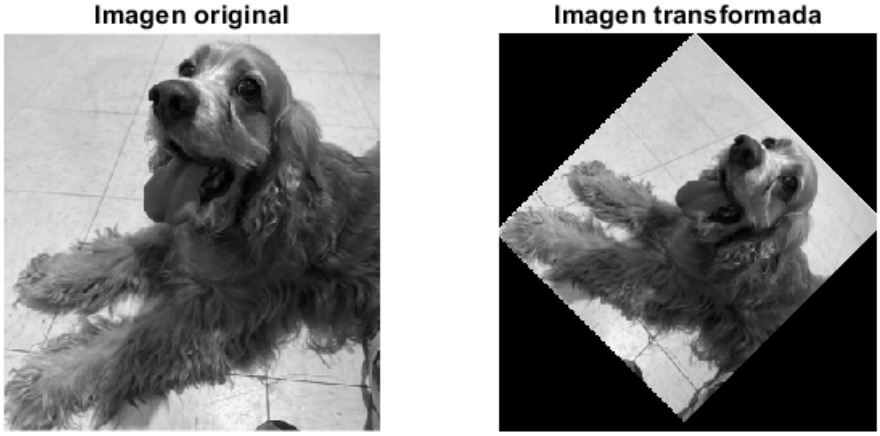

Figura 6: Imagen 2 rotada, escalada y trasladada. Dada su forma cercana a un cuadrado, es posible combinar los parámetros de tal forma que se puede realizar una rotación con una pérdida mínima de información. $T_x = 0,\ T_y = 160,\ S_{xy} = 0.71,\ \theta = 45.$

4. Conclusiones

Se logró un programa que cumple con los objetivos establecidos, aplicando transformaciones afines en imágenes siguiendo un orden establecido y tomando en cuenta los parámetros indicados. No se logró utilizar la función de interpolación bilineal de manera directa, pero se logró adaptar para su funcionamiento en un solo píxel, de manera que se pueda utilizar en este programa. El cálculo de la imagen resultante píxel por píxel es lento y computacionalmente costoso, pero aceptable para nuestros objetivos. Existen formas de optimizar el cálculo, pero esto complica la interpolación para las tonalidades resultantes.

Dado que se siguió el mapeo inverso, se necesitó elegir el tamaño de la imagen resultante antes de realizar la transformación sobre las coordenadas. Se decidió mantener el tamaño de la imagen original para simplificar la comparación, pero esto puede alterarse para evitar la pérdida de información. Por ejemplo, puede calcularse el valor que tomarán las esquinas después de la transformación, y utilizar esto para generar la imagen resultante.

Referencias

[1] Arthur Coste. Affine Transformation, Landmarks registration, Non linear Warping, October 2012.

[2] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Pearson, New York, fourth edition, global edition edition, 2017.

5. Anexo

5.1. Código

```
function new_img = affineTransform(img, tx, ty, scale, theta)
%
% AFFINETRANSFORM Applies affine transformations to input image
    with given
% parameters.
%
% NEWIMG = AFFINETRANSFORMS(img, tx, ty, scale, theta) applies
   x
% translation, y translation, scaling and rotation with given
   parameters
% in that specific order.

    % Type checking
    arguments
        img (:,:) {mustBeNumeric, mustBeNonempty}
        tx (1,1) {mustBeNumeric, mustBeFinite}
        ty (1,1) {mustBeNumeric, mustBeFinite}
        scale (1,1) {mustBeNumeric, mustBePositive}
        theta (1,1) {mustBeNumeric, mustBeFinite}
    end

    if ndims(img) ~= 2
        error('Input image must be a grayscale (2D) image.');
    end

    % Reading image size
    [height, width] = size(img);
    % Generating new image with same size
    new_img = zeros(size(img));
    % Matrix for translation on x and y axis
    translation_matrix = [1, 0, tx; 0, 1, ty; 0, 0, 1];
    % Matrix for scaling in the same factor over the x and y
       axis
    scale_matrix = [scale, 0, 0; 0, scale, 0; 0, 0, 1];
    % Matrix for rotation over the origin (top left corner)
    rot_matrix = [cosd(theta), sind(theta), 0;
        -sind(theta), cosd(theta), 0;
        0, 0, 1];

    % Transformation matrix. Translation, scaling and then
       rotation
```

```
    trans_matrix = translation_matrix * scale_matrix *
       rot_matrix;

    % Iterating over every pixel, not efficient but practical
    for row=1:height
        for col=1:width
            % Reverse transformation. We start on the output
               image, and
            % calculate position in original image. Use
               interpolation to
            % get the right color
            new_pos = [row; col; 1];
            % Position in original image. Matlab recommends
               inverted bar
            % division over the use of inv(A).
            og_pos = trans_matrix\new_pos;
            % x and y coordinates in original position. x
               indicates the
            % row for the axis used in image analysis
            x = og_pos(2);
            y = og_pos(1);
            % Interpolation. Based on assignment 2:
            % 1. Deal with positions outside of the original
               image.
            % Let it stay black and continue with the next
               pixel
            if x<0 || x > width || y<0 || y > height
                continue
            end
            % Calculate neighbor pixels in original image.
               Making sure we
            % stay inside the image boundaries
            x_low = max(floor(x), 1);
            y_low = max(floor(y), 1);
            x_top = min(x_low + 1, width);
            y_top = min(y_low+1, height);

            % Calculating four nearest pixels using coordinates
            top_left = img(y_low, x_low);
            top_right = img(y_low, x_top);
            bottom_left = img(y_top, x_low);
            bottom_right = img(y_top, x_top);

            % Needed for interpolation formula
            x_diff = x - x_low;
```

```
            y_diff = y - y_low;


            % Interpolation formula
            intensity_value = ((1 - x_diff) * (1 - y_diff) *
               top_left + ...
                     x_diff * (1 - y_diff) * top_right + ...
                     (1 - x_diff) * y_diff * bottom_left + ...
                     x_diff * y_diff * bottom_right);

            % Assigning value to pixel in new image
            new_img(row, col) = intensity_value;
        end
    end
    % Giving the right format to output image
    new_img = uint8(new_img);
end
```