

Visión por Computador

Grado en Informática Industrial y Robótica (GIROB)

Práctica 2:

Transformaciones de la imagen

Contenido:

1. Geometría Proyectiva. Nociones básicas
2. El Espacio Proyectivo P^2
3. Transformaciones proyectivas en el espacio P^2
 1. Notación vectores fila y columna
 2. Usos de las transformaciones proyectivas en P^2
4. Funciones de transformación de imágenes en PYTHON
 1. Funciones de transformación en Pillow y Matplotlib
 2. Transformación de puntos
 3. Funciones de transformación en OpenCV
5. Rectificación perspectiva
6. Entrega de resultados

Autor: José Miguel Valiente González

Bibliografía:

- Hartley R.I y Zisserman A.; **Multiple View Geometry**. Cambridge University Press. Second Edition. 2004.
- J. Howse & J. Minichino. “Learning OpenCV 4 Computer Vision with Python 3” (third Ed.) Packt Publishing Ltd. 2020.
- Official documentation for Python 3.10 <https://docs.python.org/es/3/tutorial/index.html>

Una imagen es la proyección sobre un plano de un mundo tridimensional complejo, adquirida por un sensor desde un determinado punto de vista y tras un proceso de cuantización y discretización. La imagen en sí está formada por puntos en dicho plano con unas coordenadas y un valor asociado (nivel de gris o color). Por lo tanto, hay dos tipos de operaciones que podemos realizar sobre la imagen: las que transforman las coordenadas de los puntos, como rotaciones, traslaciones, escalados; que son transformaciones puramente geométricas, y las que transforman el valor de color o gris del punto, que llamamos operaciones de procesamiento de imagen.

En esta práctica nos vamos a centrar en las **transformaciones geométricas**, para lo cual se introducen los conceptos de geometría proyectiva en el plano (2D), mediante los que se describen los operadores matemáticos para realizar estas transformaciones. En el caso 2D, esto nos permitirá realizar operaciones de modificación de la imagen, por ejemplo, para ver la misma imagen desde un ángulo diferente o para corregir distorsiones de la misma, por ejemplo, distorsión debida a la perspectiva. La geometría proyectiva 3D se utiliza para describir el modelo de la cámara y las herramientas los procesos de visión tridimensional, que se estudiarán en la segunda parte de la asignatura.

Objetivos

- Entender el fundamento matemático de las operaciones de transformación geométrica de la imagen.
- Describir las funciones que tiene Python para hacer este tipo de operaciones.

Material

- Programa Thonny y módulos **NumPy**, **Pillow**, **OpenCV** y **Matplotlib**.
- Archivos de imagen disponibles en PoliformaT. Carpeta <Imágenes VxC/Imágenes diversas>.

1. Geometría Proyectiva. Nociones básicas

Comenzamos describiendo algunos conceptos básicos de Geometría Analítica, empleando el modelo tridimensional para seguir la descripción, aunque luego seguiremos con el caso bidimensional, que se corresponde con el espacio de la imagen.

El mundo que nos rodea se puede representar mediante un **espacio afín tridimensional**, o lo que es lo mismo, un espacio euclídeo tridimensional formado por puntos. Este espacio, que se denomina $E^3 = (R^3, (x \cdot y))$, se define formalmente mediante un **espacio vectorial euclídeo** de dimensión 3 (R^3) junto con una operación sobre sus elementos denominada **producto escalar** $(x \cdot y)$. Los elementos del espacio vectorial euclídeo R^3 se denominan **VECTORES GEOMÉTRICOS** - denotados por x, y, z, \dots cursiva + negrita - y se definen como la combinación lineal de 3 vectores unitarios ortogonales $\{i, j, k\}$ - la base de dicho espacio - de forma que cada vector vendrá expresado por tres componentes:

$$x = x_1 i + x_2 j + x_3 k \quad \text{o también} \quad x = (x_1, x_2, x_3)$$

Por su parte, los elementos del espacio afín E^3 son los **PUNTOS** del espacio tridimensional real. En dicho espacio se puede definir un punto cualquiera como **Origen de Referencia**, que denominaremos **O**. Al conjunto formado por el origen de referencia y la base del espacio vectorial asociado $\{i, j, k\}$ se le denomina **Sistema Cartesiano** o **Referencia Cartesiana**, y se la representa escribiendo $(O; i, j, k)$, o también **OXYZ**.

$$p = \overrightarrow{OP} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = (p_1, p_2, p_3)^T \quad \text{Coordenadas cartesianas del punto } P \in E^3$$

2. El Espacio Proyectivo P^2

Para comprender las operaciones de transformación de la imagen es necesario introducir el concepto de espacio proyectivo. La utilidad de este tipo de espacios proviene del hecho de que podemos asociar un espacio proyectivo a un espacio euclídeo (o afín). Por ejemplo, podemos asociar (o mapear) el espacio euclídeo del plano imagen R^2 a un espacio proyectivo P^2 , o el espacio del mundo tridimensional R^3 a un espacio proyectivo P^3 . Una vez hecho esto podremos aprovechar las ventajas que ofrecen este tipo de espacios, como se verá a continuación.

Un **Espacio Proyectivo** de dimensión 2, que denominamos P^2 , se construye mediante un espacio euclídeo de dimensión 3 – ojo no 2 sino 3 - del que se ha eliminado el origen de referencia, o sea $R^3 - \{(0, 0, 0)\}$ y cuyos elementos tienen una relación de equivalencia (\sim):

$$(x_1, x_2, x_3) \sim (x'_1, x'_2, x'_3) \text{ sii}$$

$$\exists \lambda \neq 0 \text{ tal que } (x'_1, x'_2, x'_3) = \lambda (x_1, x_2, x_3)$$

En este espacio las 3-tuplas de coordenadas (x'_1, x'_2, x'_3) y (x_1, x_2, x_3) representan el mismo punto. Si lo aplicamos al espacio E^2 , se denominan **Coordenadas Homogéneas** de un punto $p = (x, y)^T \in E^2$, respecto a una referencia $(O; i, j)$, a cualquiera de las 3-tuplas de escalares (x_h, y_h, h) que satisfagan las relaciones:

$$\frac{x_h}{x} = \frac{y_h}{y} = h \quad (2)$$

En donde se adopta el criterio que cuando el denominador es cero, también lo es el numerador. 3A la tercera coordenada se la denomina **coordenada homogénea**. Se puede ‘visualizar’ el espacio proyectivo P^2 , por tratarse de un espacio de 3 dimensiones, como se muestra en la figura 1. Obsérvese que todos los puntos $p_h = h(x, y, 1)^T = (x_h, y_h, h)^T$ en el espacio proyectivo OX_hY_hh , para cualquier valor de h , representan al mismo punto $p = (x, y)^T \in E_2$ del espacio euclídeo. De la misma forma, una línea L en el plano euclídeo E^2 se representa en el espacio P^2 mediante un plano, como se muestra en la figura 1. Existe pues una dualidad (punto \leftrightarrow rayo y línea \leftrightarrow plano) entre el espacio euclídeo y su correspondiente espacio proyectivo.

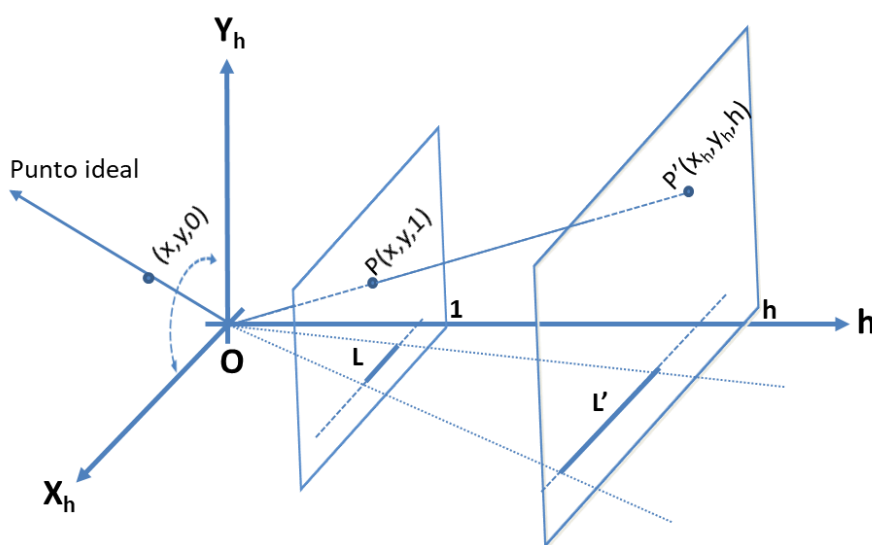


Figura 1.- Un modelo del Espacio Proyectivo P^2 .

Por simplicidad, los puntos del espacio R^2 se transforman a coordenadas homogéneas empleando la coordenada homogénea $h=1$. Si después de cualquier transformación de ese punto se obtienen unas coordenadas homogéneas (x_h, y_h, h) , se vuelve al espacio euclídeo mediante la ecuación (2).

La primera gran utilidad de la representación en coordenadas homogéneas es la posibilidad de representar los **puntos del infinito** (también llamados **puntos impropios** o **puntos ideales**), que serán todos aquellos puntos que tengan su última coordenada homogénea igual a cero. Así, el punto $(x,y,0)^T$ representa el punto del infinito en la dirección $(x,y)^T$.

Una línea L en el plano \mathbb{R}^2 se representa mediante la ecuación $ax+by+c=0$, con diferentes valores de a, b y c , para diferentes líneas en dicho espacio. Entonces, una línea se puede representar mediante la 3-tupla $(a,b,c)^T$. Sin embargo, la línea $ax+by+c=0$ es la misma que $(\lambda a)x+(\lambda b)y+(\lambda c)=0$ con lo que las tuplas $(a,b,c)^T$ y $(\lambda a, \lambda b, \lambda c)^T$ son equivalentes, para cualquier $\lambda \neq 0$.

Denominaremos a la 3-tupla $L=(a,b,c)^T$ como la representación en coordenadas homogéneas de la línea L . Un punto $P=(x,y)^T$ está en la línea L si se cumple que $ax+by+c=0$, que puede ser escrito como el producto escalar $(x,y,1)(a,b,c)=0$. En definitiva, podemos extraer las siguientes propiedades:

- Un punto x pertenece a la línea L si, y sólo si, $x^T \cdot L = 0$.
- La intersección de dos líneas L y L' es el punto $x = L \times L'$.
- La línea L que pasa por dos puntos x_1 y x_2 es $L = x_1 \times x_2$.
- La intersección de dos líneas paralelas es un punto en el infinito.

Por otro lado, los puntos en el infinito, o puntos ideales, se pueden representar mediante todos los valores $(x,y,0)$, con un punto en particular especificado por la relación $(x:y)$. Todos los puntos en el infinito forman una línea denominada **línea en el infinito**, denotada por el vector $L_\infty = (0,0,1)^T$. Se cumple pues que $(x,y,0) \cdot (0,0,1)^T = 0$.

Grados de libertad (g.d.l)

Un punto $P=(x,y)^T \in E_2$ tiene 2 grados de libertad (g.d.l), sus dos coordenadas cartesianas. El mismo punto expresado en el espacio proyectivo P^2 mediante 3 coordenadas homogéneas $P_h = (x_h, y_h, h)^T$ también tiene 2 g.d.l, pues se define completamente mediante dos relaciones $(x_h : y_h : h)$, donde $x_h/h = x$ e $y_h/h = y$.

Igualmente, una recta definida en coordenadas homogéneas como $L=(a,b,c)^T$ tiene tres valores, pero sólo dos g.d.l pues, como en el caso de los puntos, sólo es necesario conocer dos ratios ($a : b : c$). En coordenadas cartesianas esto es equivalente a decir que la recta se define por dos parámetros: la pendiente (a/b) y la ordenada en el origen (c/b).

3. Transformaciones proyectivas en el espacio P^2

La geometría proyectiva 2D es el estudio de las propiedades del plano proyectivo P^2 que son invariantes bajo un grupo de transformaciones conocidas como *proyectividades*. Una **proyectividad** es un mapeo invertible de puntos en P^2 (representados por 3 coordenadas) a puntos de P^2 que mapea líneas a líneas. Es por esto por lo que una proyectividad también se denomina como una **COLINEACIÓN** o **TRANSFORMACIÓN PROYECTIVA** u **HOMOGRAFÍA**. Las proyectividades forman un grupo ya que su inversa también es una proyectividad y la composición de dos proyectividades también lo es.

De forma general, en el espacio afín 2-dimensional E_2 es posible definir un conjunto de transformaciones lineales que representan algunas de las operaciones más usuales con los puntos de dicho espacio, como traslaciones, rotaciones, escalados etc. Estas transformaciones, que son invertibles, se representan mediante una matriz de transformación T , de dimensión 2×2 de la siguiente forma:

$$\begin{aligned} E^2 &\xrightarrow{T} E^2 \\ \forall p = \begin{pmatrix} x \\ y \end{pmatrix} \in E^2 &\quad \exists p^* = \begin{pmatrix} x^* \\ y^* \end{pmatrix} \in E^2 \quad / \quad p^* = T \cdot p \end{aligned} \quad (6)$$

De forma equivalente, en el espacio proyectivo P^2 se define una transformación proyectiva o colineación como:

$$\forall p_h = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in P^2 \quad \exists p'_h = \begin{pmatrix} x' \\ y' \\ h \end{pmatrix} \in P^2 \quad / \quad p'_h = T_h \cdot p_h \quad (7)$$

siendo T_h la **matriz de transformación generalizada** o **matriz de la colineación** para coordenadas homogéneas. Esta matriz (no singular) de dimensión 3×3 permite representar de forma condensada toda una serie de transformaciones, como se indica a continuación:

$$T_h = \begin{bmatrix} a & d & g \\ b & e & i \\ c & f & j \end{bmatrix} = \left[\begin{array}{c|c} \begin{matrix} a & c \\ b & d \end{matrix} & \begin{matrix} g \\ i \end{matrix} \\ \hline \begin{matrix} \xleftarrow{\text{Transf. Afín}} & \xleftarrow{\text{Traslación}} \\ c & j \end{matrix} & \end{array} \right] \quad (8)$$

Nótese que los vectores p_h y p'_h están definidos hasta un escalar, por lo que la matriz de la colineación (T_h) también se define hasta un factor de escala.

- Bajo una transformación proyectiva $x' = T_h \cdot x$, la línea L se transforma en L' mediante:

$$L' = T_h^{-T} \cdot L \quad \text{o bien} \quad L'^T = L^T \cdot T_h^{-1} \quad (9)$$

¿Por qué son útiles las transformaciones proyectivas?

La utilidad de esta transformación proyectiva es el hecho de que puede representar, en una sola matriz - T_h 3×3 - varias transformaciones geométricas simultáneas, como se muestra en la ecuación (8). Además, una sucesión de transformaciones se puede construir mediante pre-multiplicación de sus matrices correspondientes. Otra ventaja de trabajar en el espacio proyectivo es cuando se aplican traslaciones. En el espacio euclídeo, estas traslaciones implican sumas de vectores, que complica la concatenación de transformaciones. En cambio, en el **espacio proyectivo la concatenación de transformaciones se concreta mediante multiplicación de matrices**.

La forma de utilizar todo esto se muestra a continuación. Las coordenadas cartesianas del espacio euclídeo se pasan a homogéneas añadiendo un '1' como coordenada homogénea. A continuación se aplica la transformación proyectiva – o su composición – y finalmente se retorna al espacio euclídeo real como se muestra en la figura 2.

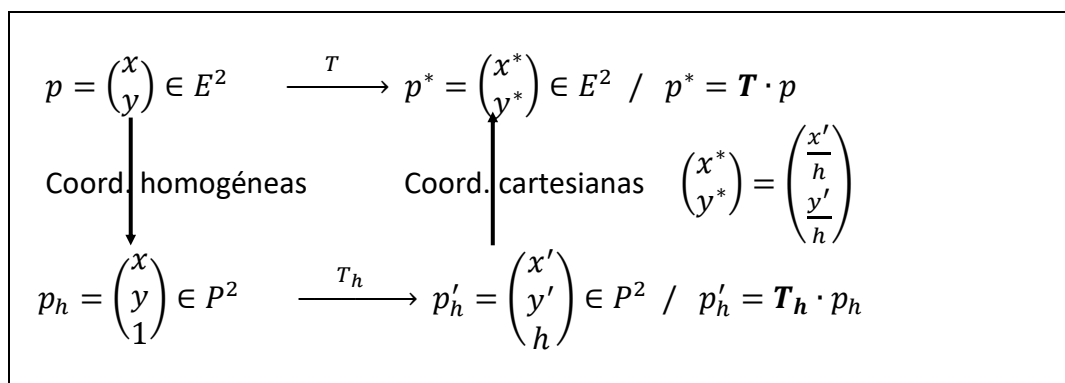


Figura 2.- Uso de la transformación proyectiva

Así mismo, es posible especializar la transformación proyectiva en 4 clases, según el tipo de efecto o distorsión geométrica que produce, definiendo una jerarquía de transformaciones, como se muestra en la Tabla 1 adjunta.


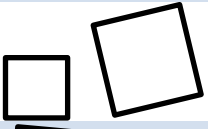


Clase	Grupo	Matriz	Distorsión	Propiedades invariantes
1	Euclídea-box (3 g.d.l)	$\begin{bmatrix} \epsilon \cos \theta & -\sin \theta & t_x \\ \epsilon \cos \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$		Longitud, área.
2	Similaridad (4 g.d.l)	$\begin{bmatrix} s \cos \theta & -s \sin \theta & t_x \\ s \cos \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix}$		Ratio de longitudes, ángulos.
3	Afín (6 g.d.l)	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A & t \\ 0^T & v \end{bmatrix}$		Paralelismo, ratio de áreas, ratio de longitudes en líneas colineales o paralelas, combinación de vectores (centroides), línea al infinito l_∞ .
4	Proyectiva (8 g.d.l)	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & v \end{bmatrix} = \begin{bmatrix} A & t \\ v^T & v \end{bmatrix}$		Concurrencia, colinealidad, orden de contacto, intersección (1 pto. contacto), tangencia (2 pto. contacto), cross-ratio

Tabla 1.- Clases de transformaciones proyectivas en 2D.

Un último detalle a considerar es cómo afectan las transformaciones anteriores a los puntos ideales o puntos en el infinito. Supongamos un punto en el infinito cualquiera $(x,y,0)$. Si aplicamos una transformación afín A el resultado es:

$$\begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix} \\ 0 \end{pmatrix} \quad (11)$$

Es decir, el punto resultante sigue siendo un punto en el infinito. Lo mismo ocurre con las isometrías o las similitudes. Sin embargo, si aplicamos una transformación proyectiva general, el resultado es:

$$\begin{bmatrix} A & t \\ v^T & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix} \\ v_1 x + v_2 y \end{pmatrix} \quad (12)$$

Siendo el resultado un punto finito. Por lo tanto, bajo una transformación proyectiva, la línea del infinito se convierte en una línea finita en el nuevo plano proyectivo.

3.1 Notación vectores fila o columna

Las expresiones de todas las ecuaciones anteriores se han realizado con **vectores columna, con pre multiplicación de la matriz T_h** .

$$p'_h = \begin{pmatrix} x' \\ y' \\ h \end{pmatrix} = \begin{bmatrix} a & d & g \\ b & e & i \\ c & f & j \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = T_h \cdot p_h \quad (13)$$

Pero también pueden expresarse con vectores fila, con post multiplicación de la matriz T_h traspuesta, de la siguiente forma:

$$p_h'^T = (x' \ y' \ h) = (x \ y \ 1) \cdot \begin{bmatrix} a & b & c \\ d & e & f \\ g & i & j \end{bmatrix} = p_h^T \cdot T_h^T \quad (14)$$

3.2 Usos de las transformaciones proyectivas en P2

La pregunta subsiguiente es ¿para qué nos sirven las transformaciones proyectivas en P^2 ? Un primer ejemplo es la **Proyección Central** que se muestra en la figura 3. Se trata de la proyección de puntos de un plano π a lo largo de un rayo o haz hasta un punto común denominado *centro de proyección*. La intersección de dichos haces con otro plano π' produce una transformación proyectiva que mapea puntos del primer plano en el segundo. Claramente las líneas en π se transforman en líneas en π' . Se trata, por definición, de una colineación o transformación proyectiva que mapea puntos de un plano a puntos de otro plano.

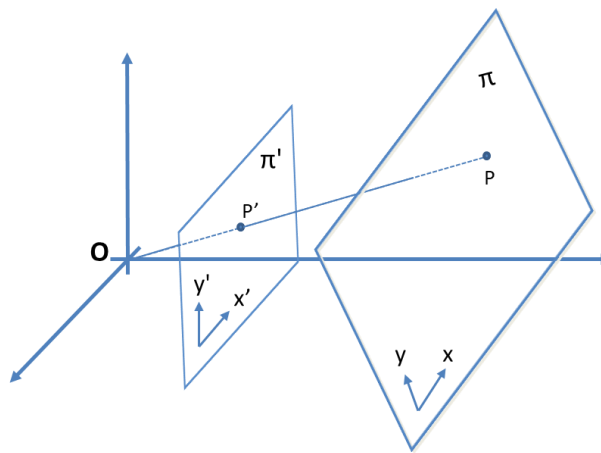


Figura 3.- Proyección Central: es una transformación proyectiva entre dos planos

La proyección central vendrá definida por una matriz de transformación proyectiva T_h - no singular de tamaño 3×3 - tal que $p' = T_h p$. Este caso se produce cuando se adquiere una imagen de un plano (π) o un objeto o parte plana del mundo real, por ejemplo, la fachada de un edificio, con una cámara (plano π' - plano del sensor de la cámara) produciéndose una distorsión debido a la perspectiva.

Si el plano del objeto es perfectamente paralelo al de la imagen entonces la distorsión es un simple escalado y/o rotación – transformación de similaridad. Si el plano del objeto está girado respecto a un eje de la cámara se producirá una transformación afín, pues se mantiene el paralelismo entre líneas. Sin embargo, si el plano del objeto tiene un giro cualquiera respecto al plano imagen entonces la transformación es totalmente proyectiva.

Es posible ‘deshacer’ esa distorsión proyectiva obteniendo la transformación inversa T_h^{-1} y aplicando dicha transformación a los puntos de la imagen. El resultado es una nueva imagen sintética en la que los objetos de dicho plano son rectificados mostrando su aspecto geométrico correcto. Véase por ejemplo la figura 4, donde se ha rectificado la imagen de una fachada de un edificio (es un objeto plano) que se ve con distorsión perspectiva.

En este caso se trata de una transformación proyectiva general (T_h tiene 8 g.d.l.) por lo cual hacen falta 4 pares de puntos correspondientes. En la imagen original se marcan 4 puntos de los vértices de una ventana. En la imagen rectificada se supone que esos cuatro puntos son los vértices de un cuadrado. Con estos 4 pares se obtiene la matriz T_h que al aplicarla a la imagen original produce la imagen rectificada.

Ojo que en este espacio proyectivo P^2 sólo se consideran las transformaciones de plano-a-plano. Si queremos describir las transformaciones desde el mundo 3D al plano de la imagen, tendremos que recurrir al espacio proyectivo P^3 , que se verá más adelante en la asignatura.

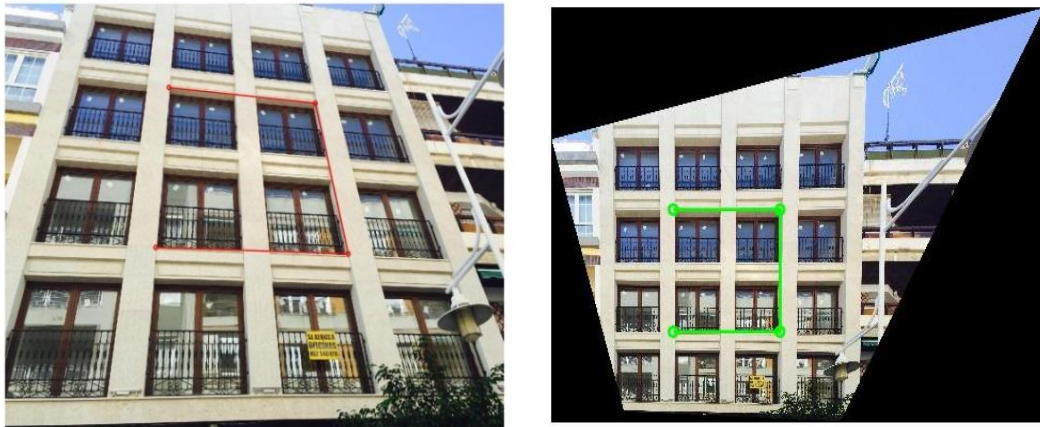


Figura 4.- Rectificación perspectiva de la imagen de un plano (fachada del edificio).

4. Funciones de transformación de imágenes en PYTHON

Como ya se indicó en la primera práctica, en Python existen distintos paquetes para trabajar con imágenes: **Pillow**, **OpenCV**, **Matplotlib**, **Scikit-Image** o incluso con **NumPy**. Para cada paquete existen clases y métodos para crear y modificar dichas imágenes. En esta práctica vamos a utilizar sólo los paquetes **Pillow**, **Matplotlib** y **OpenCV**, por ser los más habituales para estas tareas. Emplearemos el Thonny.exe como entorno de desarrollo (IDE) en esta práctica.

4.1 Funciones de transformación en Pillow y Matplotlib

Comenzaremos tomando una imagen color de prueba, por ejemplo “building4.jpg” que debemos tener en la carpeta de trabajo de Python.

```
import matplotlib.pyplot as plt
import numpy as np
import string
from PIL import Image
import math

img = Image.open("building4.jpg")
img.size      # (640, 480)

plt.figure(figsize=(9,5))
imgplot = plt.imshow(np.asarray(img))
plt.colorbar()
plt.show()
```

Utilizamos las funciones de **Matplotlib** para mostrar las imágenes, por ser muy fácil trabajar con ellas. Observamos que la imagen se lee mediante la función `Image.open()` de **PIL** (Pillow), que es de tipo `Image`. Pero para mostrarla de **Matplotlib** hay que pasarla a `ndarray` de **NumPy**, mediante `np.asarray(img)`. Con `plt.imshow()` aparece la imagen en una ventana, aunque sólo cuando se ejecuta el `plt.show()`. Si hubiera varias ventanas y varios `imshow` aparecen todas con el mismo `plt.show()`. Cuando se cierran todas estas ventanas se devuelve el control al entorno IDE.

La siguiente cuestión es cómo transformar esa imagen para hacer traslaciones, rotaciones, escalados, etc...

Para este tipo de operaciones trabajaremos en coordenadas homogéneas, como se explica en los apartados anteriores. Definiremos las matrices de transformación afín en 2D como matrices de 3x3 para los siguientes casos sencillos:

Tipo	Matriz	Tipo	Matriz
Rotación	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Traslación	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$
Escalado	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Deformación	$\begin{bmatrix} 1 & s_h & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Tabla 2 – Matrices de transformación afín 2D para **Pillow**.

La imagen se muestra en unos ejes de coordenadas estándar, donde el origen [0,0] se sitúa en la esquina superior izquierda, con el eje X hacia la derecha y el eje Y hacia abajo (invertido)

A continuación, vamos a hacer las siguientes operaciones:

- Centrar la imagen, colocando el origen [0,0] en el centro de la imagen.
- Escalar la imagen x2.
- Rotar la imagen un ángulo '*angle*' en grados.
- Posicionar el resultado en [1000,1000]

```
import matplotlib.pyplot as plt
import numpy as np
import string
from PIL import Image
import math

img = Image.open('building4.jpg')
img.size # (640, 480)

plt.figure(figsize=(9,5))
imgplot = plt.imshow(np.asarray(img))
plt.colorbar()
angle = 45
sen = math.sin(math.radians(angle))
cos = math.cos(math.radians(angle))
T_pos1000 = np.array([
    [1, 0, 1000],
    [0, 1, 1000],
    [0, 0, 1]])
# rotate - angle
T_rotate = np.array([
    [cos, -sen, 0],
    [sen, cos, 0],
    [0, 0, 1]])
# scale
T_scale = np.array([
    [2, 0, 0],
    [0, 2, 0],
    [0, 0, 1]])
# center original to image center
T_center = np.array([
    [1, 0, -img.width/2],
    [0, 1, -img.height/2],
    [0, 0, 1]])
T = T_pos1000 @ T_rotate @ T_scale @ T_center
T_inv = np.linalg.inv(T) # Image.transform uses the inverse transform matrix
img_transformed = img.transform((2000, 2000), Image.Transform.AFFINE, data=T_inv.flatten()[:6], resample=Image.NEAREST)
plt.figure(figsize=(5, 9))
plt.imshow(np.asarray(img_transformed))

plt.show()
```

Los resultados son los siguientes:

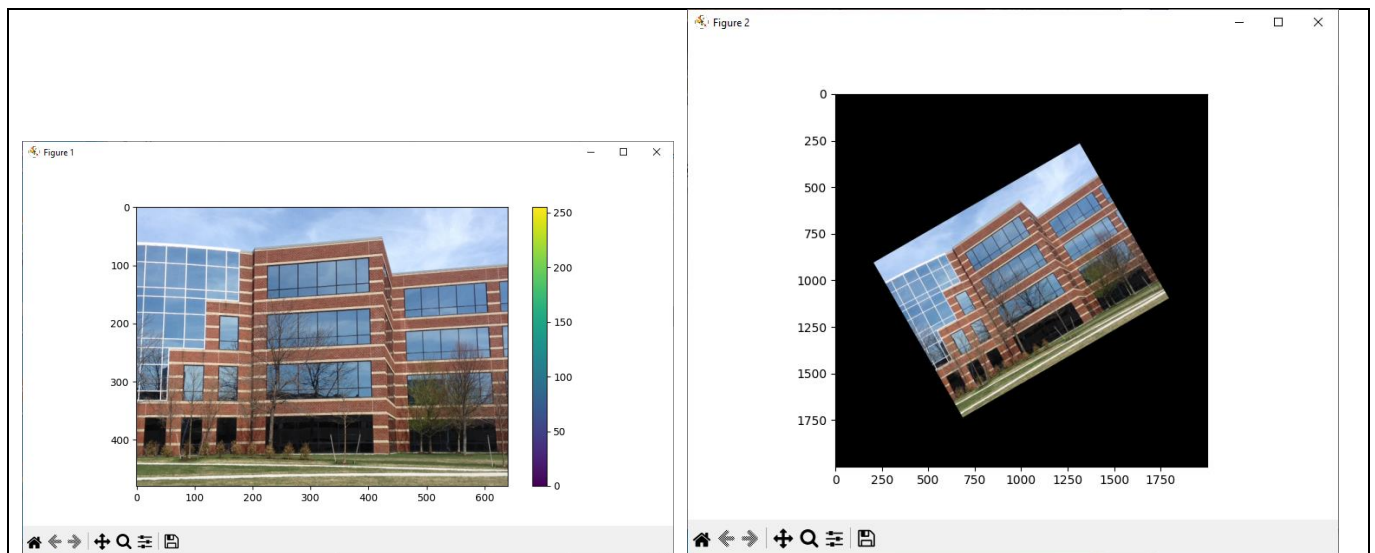


Figura 5.- Imagen original (izquierda) y resultado de la transformación afín T (derecha).

La función de transformación en **Pillow** tiene la siguiente sintaxis:

- `Image.transform(size, method, data=None, resample=Resampling.NEAREST, fill=1, fillcolor=None)`

Este método crea una nueva imagen del tamaño dado (size) y el mismo modo (mode) que el original, y copia los datos después de aplicar la transformación (data).

PARÁMETROS:

- size – Tamaño de la imagen de salida en píxeles, como una 2-tupla: (width, height).
- method – Método de transformación: `Image.Transform.EXTENT` (cut out a rectangular subregion), `Image.Transform.AFFINE` (affine transform), `Image.Transform.PERSPECTIVE` (perspective transform), `Image.Transform.QUAD` (map a quadrilateral to a rectangle), or `Image.Transform.MESH` (map a number of source quadrilaterals in one operation).
- Data – Datos extras para la transformación. Se puede incluir la INVERSA de la matriz de transformación a aplicar, pero sólo las dos primeras filas de dicha matriz.
- resample – Filtro de re-muestreo opcional: `Image.Resampling.NEAREST` (use nearest neighbour), `Image.Resampling.BILINEAR` (linear interpolation in a 2x2 environment), o `Image.Resampling.BICUBIC` (cubic spline interpolation in a 4x4 environment). Si se omite o la imagen tiene el modo "1" o "P", se establece a `Image.Resampling.NEAREST`.
- fill – Usado si el método es un objeto `ImageTransformHandler`. No usado en otro caso.
- fillcolor – Color de relleno opcional del área fuera de la transformación, en la imagen de salida p.e. 50,250,50).

IMPORTANTE:

- La matriz de transformación T puede ser la concatenación de diversas transformaciones consecutivas. En este caso se emplea la pre-multiplicación (orden T1, después T2 ,...):

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & v \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = T_4 \cdot T_3 \cdot T_2 \cdot T_1 \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad [13]$$

- La concatenación se hace mediante multiplicación de matrices, lo cual se realiza por la función `numpy.matmul`, que se simplifica mediante el operador '@'.

Ejercicio 1 – Transformaciones simples de la imagen

1. Tome una imagen cualquiera de las utilizadas en la práctica anterior, o de las disponibles en PoliformaT. Muestre la imagen en una ventana empleando el script "Afin_image.py".
2. Modifique el script anterior para aplicar una rotación de +10º a la imagen, siguiendo el ejemplo anterior, y muestre el resultado en la otra ventana, o utilice la misma ventana empleando la función `matplotlib.subplot`.
3. Cambien la matriz **T** para introducir un desplazamiento y un escalado. Varíe esos parámetros y compruebe el resultado.

Ejercicio 2 – Combinación de transformaciones de la imagen

1. Utilice el script "checkerboard.py" para crear una imagen de un tablero de ajedrez (Checkerboard) de 10x10 cuadrados de 20 pixeles de lado. Muéstrela en una imagen.
2. Cree una matriz de transformación afín T para una traslación simple $t=(t_x, t_y)=(100, 50)$. Cree una segunda matriz de transformación R para una rotación de ángulo $\Theta=30$ grados. Aplique las transformaciones indicadas a la imagen y muestre los resultados.

4.2 Transformación de puntos

Para aplicar la transformación a un punto, o grupo de puntos, utilizamos la matriz T como se muestra a continuación:

```
import matplotlib.pyplot as plt
import numpy as np
import string
import math

a, b, c, d = (0, 1, 1), (1, 0, 1), (0, -1, 1), (-1, 0, 1) # points a, b, c y d en coordenadas homogéneas
A = np.array([a, b, c, d])
T_s = np.array([[2, 0, 0], [0, 2, 0], [0, 0, 1]]) # escalado 2x
T_r = np.array([[0, 1, 0], [-1, 0, 0], [0, 0, 1]]) # rotación 90 grados
T = T_s @ T_r
# 3x3 Identity transformation matrix
I = np.eye(3)
color_lut = 'rgbc'

fig = plt.figure()
ax = plt.gca()
xs_s = []
ys_s = []
i=0
for row in A:
    output_row = T @ row # transformación de puntos
    x, y, h = row
    x_s, y_s, k = output_row
    xs_s.append(x_s)
    ys_s.append(y_s)
    c = color_lut[i]
    plt.scatter(x, y, color=c)
```

```

plt.scatter(x_s, y_s, color=c)
plt.text(x + 0.15, y, f'{string.ascii_letters[int(i)]}')
plt.text(x_s + 0.15, y_s, f'{string.ascii_letters[int(i)]}')
i+=1

xs_s.append(xs_s[0])
ys_s.append(ys_s[0])
plt.plot(x_s, y_s, color="gray", linestyle='dotted')
plt.plot(xs_s, ys_s, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-10, 10, 1.0))
ax.set_yticks(np.arange(-10, 10, 1.0))
plt.grid()
plt.show()

```

Ejercicio 3 – Transformaciones de puntos

1. Escriba el script anterior en un archivo “RotacionPuntos.py”
2. Cambie la matriz de transformación afín T para que produzca una traslación simple $t=(t_x, t_y)=(-3, 3)$. Cree una segunda matriz de transformación R para una rotación de ángulo $\Theta=30$ grados. Aplique las transformaciones indicadas a los puntos y muestre los resultados, que deben ser como la siguiente figura:

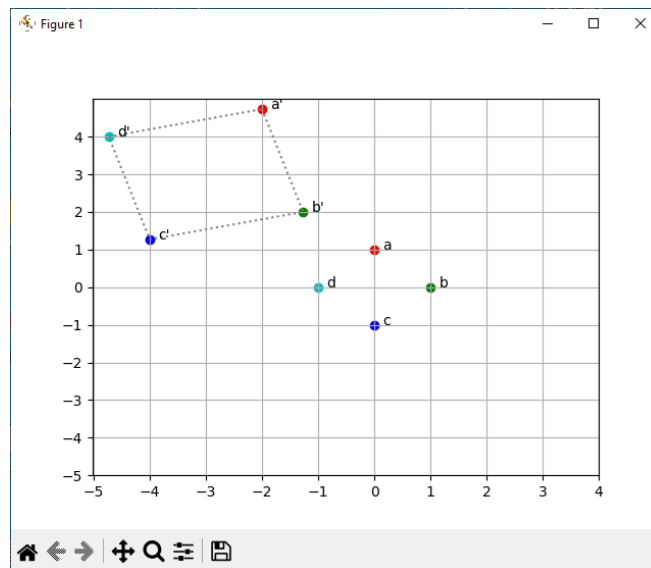


Figura 6.- Resultado de la transformación de puntos con **Pillow**.

4.3 Funciones de transformación en OpenCV

OpenCV es una biblioteca de funciones muy completa, que permite leer, escribir y cambiar de tipo las imágenes que se obtienen en formato **Mat**. También se pueden visualizar estas imágenes mediante `cv2.imshow()`. A diferencia de **Pillow**, las ventanas creadas sólo se cierran mediante `cv2.destroyAllWindows()`, pero para esperar se puede emplear `cv2.waitKey()`. A continuación, vemos un ejemplo para hacer esto:

```

import cv2
import numpy as np
import matplotlib.pyplot as plt

src = cv2.imread('building4.jpg')
dst = cv2.resize(src, (256, 256), interpolation=cv2.INTER_CUBIC)

cv2.imshow('Imagen original', src)
cv2.imshow('Imagen escalada', dst)

```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Las funciones de visualización de *matplotlib.pyplot*, como `plt.imshow()`, también se pueden usar directamente con las imágenes de tipo `Mat`, poniendo:

```
plt.subplot(121),plt.imshow(src),plt.title('Input') # Visualización en matplotlib.pyplot
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

Ejercicio 4 – Visualización de OpenCV empleando Matplotlib

1. Cree un archivo “VisualizaOpenCV.py” y escriba el script anterior, añadiendo las instrucciones indicadas. Compruebe si se visualizan bien las imágenes con Matplotlib. ¿Qué problema se plantea?

Pista: Las imágenes en OpenCV tienen el formato ‘BGR’, mientras que Matplotlib utiliza el formato ‘RGB’. Busque en OpenCV una función para hacer esa conversión antes de visualizar.

En cuanto a las funciones de transformación geométrica de **OpenCV**, se pueden clasificar en las siguientes:

1. Transformaciones afines predefinidas.
 - `cv2.resize`, `cv2.flip`, `cv2.Mirror`, `cv2.2DRotationMatrix`
2. Transformaciones afines genéricas.
 - `cv2.warpAffine`, `cv2.getAffineTransform`
3. Transformaciones perspectivas.
 - `cv2.warpPerspective`, `cv2.getPerspectiveTransform`
4. Transformaciones de mapeo arbitrario.
 - `cv2.Remap`

La función de **escalado** (resize) es la siguiente:

- `Mat dst = cv2.resize (Mat src, size, int inter=CV_INTER_LINEAR)`

Este método crea una nueva imagen del tamaño dado (size) y el mismo modo (mode) que el original, y copia los datos después de aplicar la transformación (data).

PARÁMETROS:

- `src`: Imagen fuente a re-escalar.
- `size` – Tamaño de la imagen de salida en píxeles, como una 2-tupla: (width, height), o factores de escalado del eje X (fx) y del eje Y (fy)
- `interpolation` – Método de interpolación: `cv2.INTER_AREA`; `cv2.INTER_LINEAR`; `cv2.INTER_CUBIC`.

Ejemplo:

```
res = cv2.resize(img,None,fx=2,fy=2,interpolation = cv.INTER_CUBIC)
#OR
height,width = img.shape[:2]
res = cv2.resize(img,(2*width,2*height),interpolation = cv.INTER_CUBIC)
```

Por otro lado, las operaciones de traslación, rotación o deformación se realizan mediante la función `cv.warpAffine()`, cuya sintaxis en Python es la siguiente:

- `Mat dst = cv2.warpAffine(Mat src, M, dsize[, dst[, flags [, borderMode[, borderValue]]]])`

PARÁMETROS:

- **src**: Imagen fuente a transformar.
- **dst**: Imagen de salida, de tamaño dsize y el mismo tipo que la imagen fuente.
- **M**: Matriz de transformación 2x3
- **dsize**: Tamaño de la imagen de salida.
- **flags**: Combinación de métodos de interpolación-
- **borderMode**: Método de extrapolación de píxeles. `cv2. BORDER_CONSTANT`, etc..
- **borderValue**: En el caso de bordes constantes, es el valor a usar.

La matriz de transformación M es de tamaño 2x3. Se trata de la misma matriz T vista en la teoría, excepto que la tercera fila no se utiliza, pues no produce un resultado útil a la salida (Tabla 2).

Tipo	Matriz	Tipo	Matriz
Rotación	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \end{bmatrix}$	Traslación	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$
Escalado	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix}$	Deformación	$\begin{bmatrix} 1 & s_h & 0 \\ s_v & 1 & 0 \end{bmatrix}$

Tabla 3 – Matrices de transformación afín 2D para **OpenCV**.

Estas matrices se pueden definir directamente, o bien empleando alguna función auxiliar, como:

- `Mat M = cv2.getRotationMatrix2D(center, angle, scale)`

```
import cv2
import numpy as np

src = cv2.imread('building4.jpg')
rows, cols = src.shape[:2]

M = np.float32([[1, 0, 100], [0, 1, 50]])
dst = cv2.warpAffine(src, M, (rows + 100, cols + 50))

cv2.imshow('Imagen original', src)
cv2.imshow('Imagen trasladada', dst)

M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
dst2 = cv2.warpAffine(src, M, (cols, rows))

cv2.imshow('Imagen rotada', dst2)
cv2.waitKey()
cv2.destroyAllWindows()
```

Ejercicio 5 – Combinación de transformaciones simples en OpenCV

- Repita el ejercicio 1, de concatenación de transformaciones simples de la imagen, pero utilizando las funciones de **OpenCV** antes indicadas. Para esto cree el script “warpAffineOpenCV.py”

Cuando tratamos de hacer transformaciones geométricas de tipo proyectivo, las matrices de transformación son de tamaño 3x3, y coinciden con las indicadas en la Tabla1. En este caso, en vez de emplear `cv2.warpAffine()` hay que emplear `cv2.warpPerspective()`, con la siguiente sintaxis:

- `Mat dst = cv2.warpPerspective(Mat src, T, dsize[, dst[, flags [, borderMode[, borderValue]]]])`

Los parámetros son los mismos que en la proyección afín, excepto la matriz T que ahora es de 3x3. En tal caso también se pueden incluir este tipo de transformaciones con la tercera línea de esa matriz.

5. Rectificación perspectiva

Como se indicó en el apartado 3.2, se pueden emplear las funciones de transformación para deshacer o rectificar el efecto de la perspectiva en el plano 2D. El ejemplo se muestra en la figura 4, donde una fachada de un edificio (que es un plano) ha quedado deformada por la perspectiva al tomar la foto. Se puede deshacer esta distorsión indicando cómo se puede transformar dicha imagen para que los puntos distorsionados (que supuestamente forman un cuadrado) sean transformados a un cuadrado perfecto.

Pero, para hacer esto no podemos emplear las funciones `cv2.warpAffine()` y `cv2.warpPerspective()` antes indicadas, pues éstas una transformación conocida (matriz T). Para hacer lo indicado antes hay que emplear nuevas funciones, que nos permiten conocer qué matriz de transformación hay que imponer para que un conjunto de puntos (3 o 4) se conviertan en otro conjunto de puntos de salida. Estas funciones son:

- Mat M = `cv2.getAffineTransform(src, dst)` # 3 pares de puntos
- Mat T = `cv2.getPerspectiveTransform(src, dst[, solveMethod])` # 4 pares de puntos

PARÁMETROS:

- **src**: Coordenadas de triángulo (3 puntos) o cuadrángulo (4 punto) en la imagen fuente a transformar.
- **dst**: Coordenadas del correspondiente triángulo o cuadrángulo en la imagen de destino.
- **solveMethod**: Método de cálculo: `cv2.DECOMP_LU`, `cv2.DECOMP_SVD`, `cv2.DECOMP_EIG`, `cv2.DECOMP_CHOLESKY`, `cv2.DECOMP_QR`, `cv2.DECOMP_NORMAL`.

Para marcar los puntos podemos emplear el ratón. En OpenCV la captura de los eventos del ratón se realiza mediante `cv2.setMouseCallback()`, indicando una función a ejecutar con cada click. Asimismo, para marcar los puntos o las líneas en la imagen se pueden emplear las funciones de dibujo de **OpenCV**, tales como `cv2.line()`, `cv2.rectangle()`, `cv2.circle()`, `cv2.polylines()`, `cv2.drawMarker()`, `cv2.putText()`, etc. Estas funciones se pueden encontrar en:

https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html

A modo de ejemplo véase el código disponible en el archivo `utils.py`, que incorpora una función `ginput()`, similar a la misma de Matlab. Esta función permite marcar 'n' puntos sobre la imagen y devuelve la lista de dichos puntos. El último punto es siempre igual al primero, para cerrar el conjunto. Pruebe este código que emplea dicha función:

siguiente:

```
import cv2
from utils import ginput

# read image
img = cv2.imread('building5.jpg')
# show image
cv2.imshow('image', img)
# graphic input of 5 points
pts = ginput('image', img, 6)
# print the result
print(pts)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


► Ejercicio 6 – Rectificación perspectiva de una imagen

En un nuevo script, llamado 'rectifica.py', realice las siguientes acciones:

1. Tome la imagen de la fachada de un edificio cualquiera (Buildinmg4.jpg, Torredelmar.jpg, Santa-Maria-Micaela-2.jpg, edificio-caledonia-zona-16-3.jpg,) y escriba un código para marcar – empleando `ginput()` – 4 puntos de control correspondientes a una zona supuestamente cuadrada – p.e. una ventana o un grupo de ventanas, etc.. - y utilice la función `getPerspectiveTransform()` para obtener la transformación geométrica correspondiente a un segundo grupo de puntos de control (`outs`) que forman un cuadrado de lado 200.

```
.....
pts = ginput('image', src, 5)
print(pts)
outs = [[0,0], [200,0], [200,200], [0,200]] + pts[0]
outs = np.array(outs,np.float32)
print(outs)
M = cv2.getPerspectiveTransform(pts,outs)
.....
```

2. Rectifique la imagen original utilizando la matriz de transformación obtenida en el apartado anterior y la función `cv2.warpPerspective()`.
3. Muestre una segunda imagen con la imagen rectificada y el polígono indicado por los puntos de control.
4. Repita lo anterior con otro edificio para comprobar el comportamiento de la rectificación perspectiva. En caso necesario, modifique el tamaño de la imagen original para que sea más manejable.

► Ejercicio 7 – Transformación de espacios de color

1. Tomar una imagen RGB y muestre los tres componentes R (rojo), G (verde) y B (azul) como imágenes de gris por separado. Pruébalo con la imagen 'AloeVera.jpg'.
2. Transforme la misma imagen a HSV y muestre los tres componentes H (hue), S (Saturation) y V (Value) como imágenes en gris por separado.
3. Ídem que lo anterior, pero a espacio Lab.

6. Entrega de resultados

Para entregar los resultados de la práctica, haga un documento (Word o similar) con los distintos ejercicios incluyendo, en cada uno, lo siguiente:

- Enunciado del ejercicio
- Script de Python (copy – paste)
- Imágenes de los resultados (menú Edit->Copy figure en la ventana y paste en el Word)
- Comentarios personales sobre el ejercicio y los resultados (si procede)

Incluya, en la página inicial, el título de la práctica y el nombre del/los alumnos implicados. Convierta el archivo a pdf y súbalo a PoliformaT al espacio compartido de cada alumno implicado.