Doble Grado en Ingeniería Informática y Matemáticas

VISIÓN POR COMPUTADOR
(E. Computación y Sistemas Inteligentes)

PROYECTO FINAL:

Implementar la creación de panoramas lineales con proyección en superficies cilíndricas o esféricas y mezcla de colores usando el algoritmo de Burt-Adelson.

UNIVERSIDAD DE GRANADA

Alberto Estepa Fernández
Email: albertoestep@correo.ugr.es

Carlos Santiago Sánchez Muñoz
Email: carlossamu7@correo.ugr.es

8 de enero de 2020

Índice

1. Introducción

En el marco de la Visión por Computador encontramos la construcción de panoramas lineales. Esto consiste en la creación de imágenes panorámicas a partir de un conjunto de imágenes de un mismo lugar. Este tema tiene una profundidad y utilidad muy grande en diferentes ámbitos y nos abre un mundo dentro de la Visión por Computador.

La necesidad de combinar dos o más imágenes en un mosaico más grande ha surgido en diferentes contextos. Sin ir más lejos para fotos tomadas en el espacio, donde aparecen de estrellas y satélites que debido a sus órbitas sólo se pueden apreciar en diferentes puntos. Aquí la construcción de vistas panorámicas a partir de múltiples fotografías permite conseguir un campo de visión y nivel de detalle mucho mayor. Por ejemplo, a menudo fotografías de *Landsat* se ensamblan en vistas panorámicas de la Tierra.

Problema: nuestro problema a estudiar es el acople de las imágenes. Sabemos de la práctica 3 y lo estudiado en la teoría que este acople puede ser no satisfactorio ya que al tomar dos fotos de un mismo sitio percibimos diferencias de luces, sombras, brillos, contrastes y otros. A la hora de la creación de la vista panorámica esto se nota en la unión de las imágenes.

Solución propuesta: Vamos a usar el Algoritmo de Burt Adelson para la mezcla en la creación de la panorámica. Para ello vamos a hacer uso de dos proyecciones: proyección cilíndrica y proyección esférica.

Vamos a usar lo aprendido a lo largo del desarrollo de la asignatura pues necesitaremos filtros, construcciones de pirámides laplacianas, restauración de pirámides laplacianas, cálculo de correspondencias, cálculo de homografías más los nuevos contenidos ya mencionados.

La diferencia entre lo que se va a realizar ahora y los mosaicos que trabajamos en la práctica 3 es que al usar las proyecciones y al hacer la mezcla con el algoritmo de Burt Adelson conseguimos un suavizado en la unión de las imágenes y una construcción que parece prometedora.

El material donde encontramos detallada la explicación de Burt Adelson lo encontramos en el siguiente link. Dicho material es apoyo fundamental de lo que aquí se realiza y explica.

`http://www.cs.princeton.edu/courses/archive/fall05/cos429/papers/burt_adelson.pdf`

2. Proyección cilíndrica

La proyección cilíndrica consiste en recorrer todos los píxeles de la imagen (y, x) y calcular su proyección cilíndrica (y', x') bajo la aplicación matemática que describe este fenómeno. Para ello necesitamos considerar el centro de la imagen (c_y, c_x).

Para la transformación necesitamos dos parámetros:

- la distancia focal, f, que es la distancia desde el centro óptico
- y el factor de escalado, s, que define el escalado de la imagen para dibujarla en la original.

La fórmula para la proyección explicada es la siguiente:

$$x' = s * \arctan\left(\frac{x - c_x}{f}\right) + c_x \tag{1}$$

$$y' = s * \frac{y - c_y}{\sqrt{(x + c_x)^2 + f^2}} + c_y \tag{2}$$

El procedimiento es crear una nueva imagen en donde en el píxel (y', x') guardamos el valor que hay en el píxel (y, x) de la imagen original. Este valor puede ser una terna RGB en caso de que sea una imagen tribanda. Esto es fácil de evitar pues hemos programado esta función de manera recursiva y una vez que tenemos la proyección implementada para un canal es fácil extenderlo a tres usando las funciones `cv2.split()` y `cv2.merge()`.

Respecto a los parámetros distancia focal y factor de escala, la imagen tendrá mayor proyección cuanto más pequeños sean estos parámetros. Recíprocamente para valores grandes será más similar a la original pues tendrá menos proyección.

Vamos a trasladar todo lo explicado a código. Programamos en Python una función `cylindricalProjection()` a la cual le pasamos la imagen y los parámetros y nos devuelve la proyección cilíndrica de esa imagen. Veamos:

```
""" Proyeccion cilindrica de una imagen
- img: imagen a proyectar.
- f: distancia focal.
- s: factor de escalado.
"""
def cylindricalProjection(img, f, s):
    if len(img.shape) == 3:
        # Si está en color separamos en los distintos canales
        canals = cv2.split(img)
        canals_proy = []
        for n in range(len(canals)):
            # Proyección cilindrica
            canals_proy.append(cylindricalProjection(canals[n],f,s))
        # Mezclamos canales
        proyected=cv2.merge(canals_proy)

    else:
```

```
    proyected = np.zeros(img.shape) # Imagen proyectada
    y_center = img.shape[0]/2       # centro y
    x_center = img.shape[1]/2       # centro x
    # Proyectamos la imagen
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            y_proy = floor(s * ((i-y_center) / np.sqrt((j-x_center)*(j-x_center) + f*f))
                     + y_center)
            x_proy = floor(s * np.arctan((j-x_center) / f) + x_center)
            proyected[y_proy][x_proy] = img[i][j]
    # Normalizamos
    proyected = cv2.normalize(proyected, proyected, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

return proyected
```

Ahora vamos a probarlo sobre diferentes imágenes. La primera imagen que vamos a proyectar es del palacio de Carlos V:

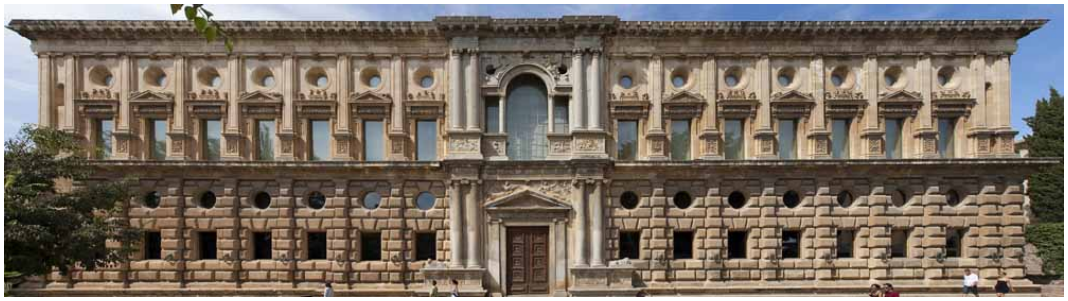

Imagen 1: Palacio de Carlos V

La primera proyección la hacemos con valores distancia focal y factor de escalado de 600. El resultado queda así:

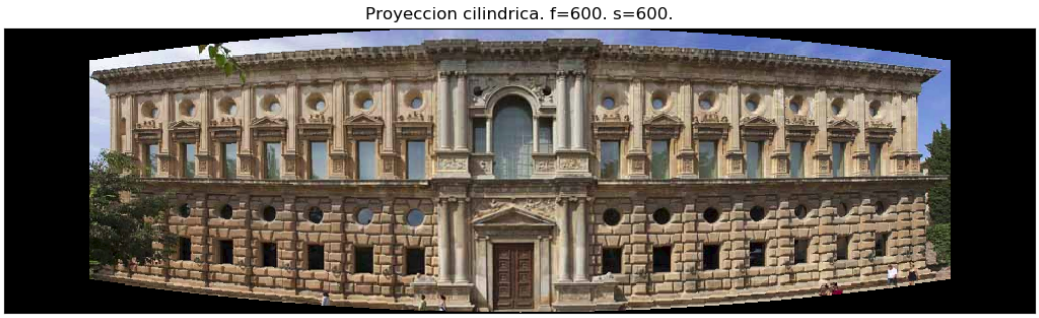

Imagen 2: Palacio de Carlos V proyectado cilíndricamente con $f = 600$ y $s = 600$

La segunda proyección la hacemos con valores distancia focal y factor de escalado de 900. El resultado queda así:

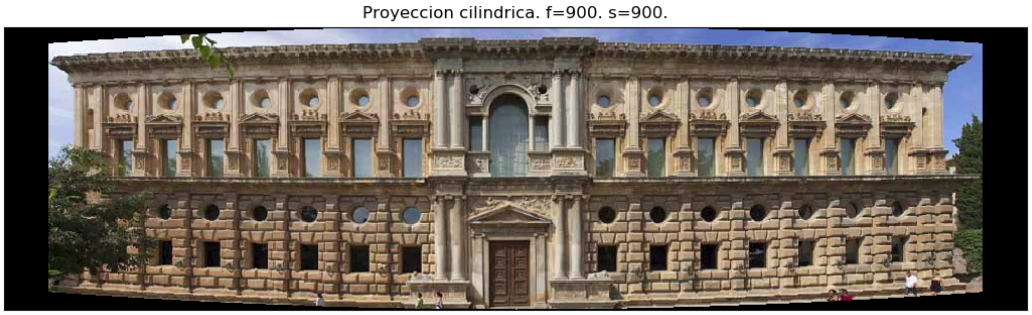

Imagen 3: Palacio de Carlos V proyectado cilíndricamente con $f = 900$ y $s = 900$

Haciendo una valoración de los resultados lo primero que debemos de darnos cuenta es lo que mencionamos anteriormente de que cuanto mayores sean los parámetros f y s más se parece a la imagen original. La *Imagen 3* se parece a la original más que la otra, la cual está más proyectada.

Todas las imágenes tienen las mismas dimensiones, ello nos permite observar que fruto de la proyección hay una zona de la imagen (las esquinas) que se queda en negro. La explicación es sencilla, nosotros antes de proyectar rellenamos la matriz con ceros y no existen píxeles (y, x) que viajen a esos de las esquinas por medio de la proyección.

La segunda imagen que vamos a proyectar es una foto de la Alhambra vista desde el Albaicín granadino que será parte de la futura vista panorámica:

Imagen 4: Alhambra

La primera proyección la hacemos con valores distancia focal y factor de escalado de 600. La segunda proyección la hacemos con valores distancia focal y factor de escalado de 800. El resultado queda así:

Imagen 5: Alhambra proyectada cilíndricamente con $f = 600$ y $s = 600$

Imagen 6: Alhambra proyectada cilíndricamente con $f = 800$ y $s = 800$

Al igual que antes la *Imagen 6* es la que más parecida a la original y la que tiene mayor distancia focal y factor de escalado.

La que está más proyectada, la *Imagen 5*, es la que deja más esquina en color negro.

3. Proyección esférica

Para la proyección esférica la idea es la misma, solo que empleando su fórmula correspondiente, el procedimiento e implementación es idéntico, únicamente variaremos el cómputo de la misma debido a que presenta una transformación distinta:AAAAAAA

Para la proyección esférica también vamos a recorrer todos los píxeles de la imagen (y, x) calculando su proyección cilíndrica (y', x') bajo la aplicación matemática que describe dicho fenómeno. Para ello necesitamos considerar el centro de la imagen (c_y, c_x).

Para la transformación necesitamos dos parámetros:

- la distancia focal, f, que es la distancia desde el centro óptico
- y el factor de escalado, s, que define el escalado de la imagen para dibujarla en la original.

La fórmula para la proyección explicada es la siguiente:

$$x' = s * \arctan\left(\frac{x - c_x}{f}\right) + c_x \tag{3}$$

$$y' = s * \arctan\left(\frac{y - c_y}{\sqrt{(x + c_x)^2 + f^2}}\right) + c_y \tag{4}$$

Al igual que antes el procedimiento es crear una nueva imagen en donde en el píxel (y', x') guardamos el valor que hay en el píxel (y, x) de la imagen original. Este valor puede ser una terna RGB en caso de que sea una imagen tribanda y lo evitamos usando recursividad ya que una vez que tenemos la proyección implementada para un canal es fácil extenderlo a tres usando las funciones `cv2.split()` y `cv2.merge()`.

Tenemos otra vez la conclusión de que la imagen tendrá mayor proyección cuanto más pequeños sean los parámetros distancia focal y factor de escala la imagen tiene mayor proyección y recíprocamente para valores grandes será más similar a la original.

Vamos a trasladar todo lo explicado a código. Programamos en Python una función `sphericalProjection()` a la cual le pasamos la imagen y los parámetros y nos devuelve la proyección cilíndrica de esa imagen. Veamos:

```
""" Proyeccion éesfrica de una imagen
- img: imagen a proyectar.
- f: distancia focal.
- s: factor de escalado.
"""
def sphericalProjection(img, f, s):
    if len(img.shape) == 3:
        # Si está en color separamos en los distintos canales
```

```
        canals = cv2.split(img)
        canals_proy = []
        for n in range(len(canals)):
            # Proyección esférica
            canals_proy.append(sphericalProjection(canals[n],f,s))
        # Mezclamos canales
        proyected=cv2.merge(canals_proy)

    else:
        proyected = np.zeros(img.shape) # Imagen proyectada
        y_center = img.shape[0]/2       # centro y
        x_center = img.shape[1]/2       # centro x
        # Proyectamos la imagen
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                y_proy = floor(s * np.arctan((i-y_center) / np.sqrt((j-x_center)*(j-x_center)+f*f)) +
                x_proy = floor(s * np.arctan((j-x_center) / f) + x_center)
                proyected[y_proy][x_proy] = img[i][j]
        # Normalizamos al tipo uint8
        proyected = cv2.normalize(proyected, proyected, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

    return proyected
```

Ahora vamos a probarlo sobre diferentes imágenes. La primera como ya sabemos es del palacio de Carlos V. Realizamos una primera proyección con valores distancia focal y factor de escalado de 600. El resultado queda así:

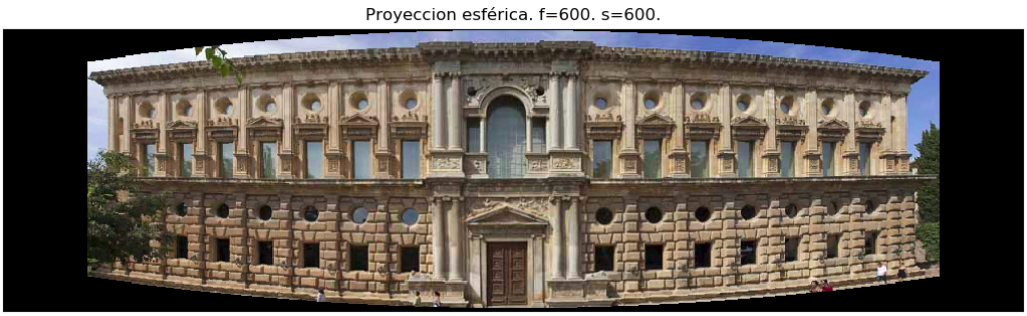

Imagen 7: Palacio de Carlos V proyectado esféricamente con $f = 600$ y $s = 600$

Ahora hacemos otra proyección con valores distancia focal y factor de escalado de 900. El resultado queda así:

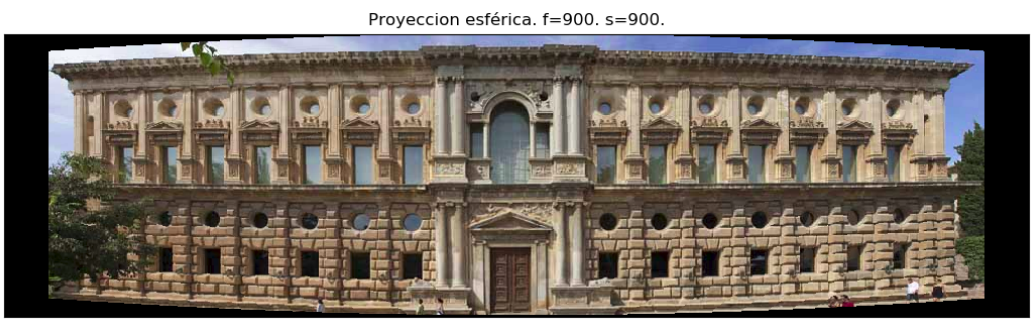

Imagen 8: Palacio de Carlos V proyectado esféricamente con $f = 900$ y $s = 900$

Es momento de valorar los resultados de esta proyección. Confirmamos lo que ya mencionamos anteriormente de que cuanto mayores sean los parámetros f y s más se parece a la imagen original. La *Imagen 8* es más parecida a la original y la *Imagen 7* tiene una mayor proyección. De nuevo se nos quedan zonas de la imagen (las esquinas) en negro.

La segunda imagen que vamos a proyectar es la foto de la Alhambra. Para la primera proyección usamos parámetros distancia focal y factor de escalado de 600. La segunda proyección la hacemos con valores distancia focal y factor de escalado de 800. Mostramos el resultado:

Imagen 9: Alhambra proyectada esféricamente con $f = 600$ y $s = 600$

Imagen 10: Alhambra proyectada esféricamente con $f = 800$ y $s = 800$

Al igual que antes la *Imagen 10* es la que más parecida a la original y la que tiene mayor distancia focal y factor de escalado. La *Imagen 9* está más proyectada y deja más esquina en color negro.

4. Obtención de correspondencias y construcción del mosaico

5. Algoritmo de Burt-Adelson

6. Resultados

7. Conclusiones y trabajos futuros

```
def criterioHarris(eigenVal1, eigenVal2, threshold):
```