

SISTEMAS DE PERCEPCIÓN

Ejercicio práctico 2: Eliminación de distorsión en imagen

Álvaro Calvo Matos
Damián Jesús Pérez Morales

Índice

1. Introducción	2
2. Planteamiento del problema	2
3. Imagen 1. Biconvexa	5
4. Imagen 2. Cóncava	6
5. Código implementado en MATLAB	7
5.1 Script ej_pract2.m	7
5.2 Script corrige.m	8
6. Conclusiones.....	11

1. INTRODUCCIÓN

En el segundo ejercicio práctico propuesto, se pretende trabajar con imágenes que están de partida distorsionadas para luego eliminar dicha distorsión mediante el uso de interpoladores. En este caso, se empleará el interpolador del vecino más cercano (*nearest neighbour*) y el interpolador bilineal, analizando las diferentes imágenes generadas a partir de cada interpolador.

En concreto, se trabajarán con las dos imágenes siguientes, que emplean diferentes lentes, siendo la imagen 1 y la imagen 2 distorsionadas respectivamente:

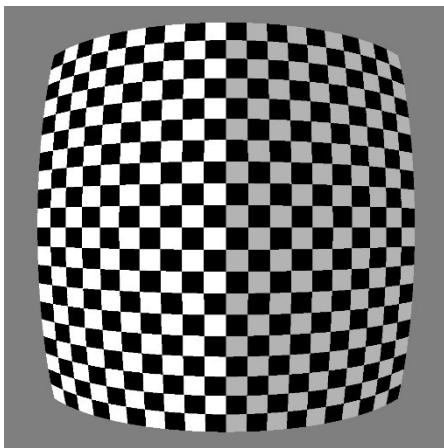


Ilustración 1. Imagen distorsionada 1 -
Biconvexa

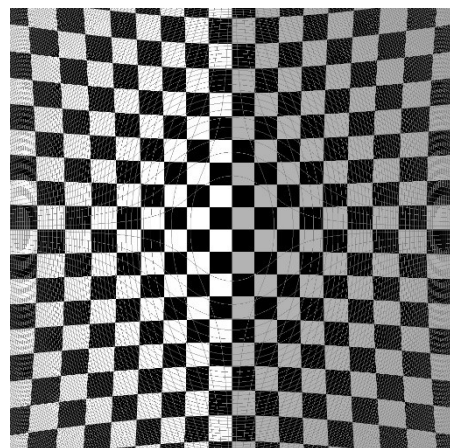


Ilustración 2. Imagen distorsionada 2 -
Cóncava

2. PLANTEAMIENTO DEL PROBLEMA

Para resolver este ejercicio de la forma más sensata, se tiene que considerar que sabemos de qué imagen se parte (imagen NO distorsionada) y, a partir de la imagen distorsionada (la que se tiene), aplicar una serie de cuentas para eliminar la distorsión. Dichas cuentas son las siguientes:

- Para el interpolador del vecino más cercano:

$$g(i,j) = \text{round} (f(i,j))$$

- $g(i,j)$ → imagen resultante (no distorsionada)
- $f(i,j)$ → imagen de partida (distorsionada)

- Para el interpolador bilineal (ponderando por **proximidad**):
El procedimiento a seguir es:
 1. Obtener las coordenadas de u y de v normalizadas
 2. Analizar la primera fila v_{d_1} , considerando que u_{d_1} es el píxel más cercano hacia la izquierda respecto a u_d y u_{d_2} es el píxel más cercano hacia la derecha respecto a u_d .
 3. Analizar la segunda fila v_{d_2} , considerando que u_{d_1} es el píxel más cercano hacia la izquierda respecto a u_d y u_{d_2} es el píxel más cercano hacia la derecha respecto a u_d .
 4. Combinar ambas filas.

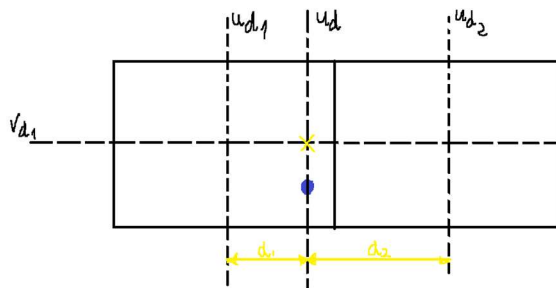


Ilustración 3. Planteamiento fila 1

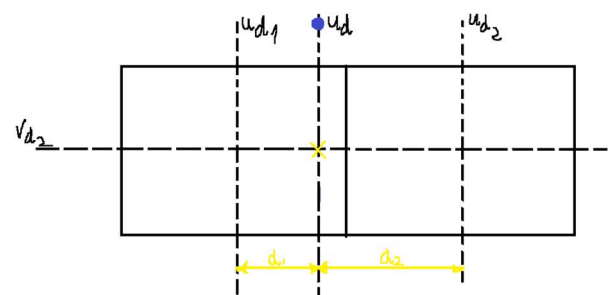


Ilustración 4. Planteamiento fila 2

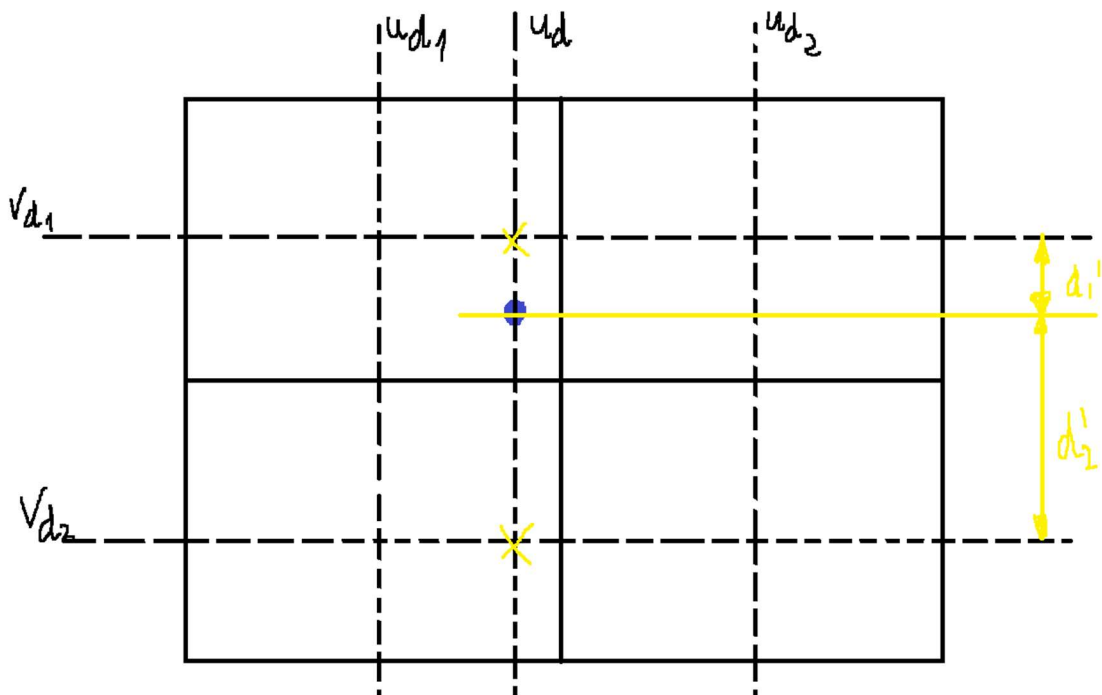


Ilustración 6. Planteamiento combinación de filas

- Primera fila:

$$I_{pix}(v_{d_1}, u_d) = \frac{u_{d_2} - u_d}{u_{d_2} - u_{d_1}} * I_{pix}(v_{d_1}, u_{d_1}) + \frac{u_d - u_{d_1}}{u_{d_2} - u_{d_1}} * I_{pix}(v_{d_1}, u_{d_2})$$

- $\frac{u_{d_2} - u_d}{u_{d_2} - u_{d_1}} = d_2$ (ilustración)
- $\frac{u_d - u_{d_1}}{u_{d_2} - u_{d_1}} = d_1$ (ilustración)
- $I_{pix}(v_{d_1}, u_{d_1})$ = intensidad de píxel de la imagen de partida del píxel (v_{d_1}, u_{d_1})
- $I_{pix}(v_{d_1}, u_{d_2})$ = intensidad de píxel de la imagen de partida del píxel (v_{d_1}, u_{d_2})

○ Segunda fila:

$$I_{pix}(v_{d_2}, u_d) = \frac{u_{d_2} - u_d}{u_{d_2} - u_{d_1}} * I_{pix}(v_{d_2}, u_{d_1}) + \frac{u_d - u_{d_1}}{u_{d_2} - u_{d_1}} * I_{pix}(v_{d_2}, u_{d_2})$$

- $I_{pix}(v_{d_2}, u_{d_1})$ = intensidad de píxel de la imagen de partida del píxel (v_{d_2}, u_{d_1})
- $I_{pix}(v_{d_2}, u_{d_2})$ = intensidad de píxel de la imagen de partida del píxel (v_{d_2}, u_{d_2})

○ Combinando ambas filas:

$$I_{pix}(v_d, u_d) = \frac{v_{d_2} - v_d}{v_{d_2} - v_{d_1}} * I_{pix}(v_{d_1}, u_d) + \frac{v_d - v_{d_1}}{v_{d_2} - v_{d_1}} * I_{pix}(v_{d_2}, u_d)$$

- $\frac{v_{d_2} - v_d}{v_{d_2} - v_{d_1}} = d_2'$ (ilustración)
- $\frac{v_d - v_{d_1}}{v_{d_2} - v_{d_1}} = d_1'$ (ilustración)
- $I_{pix}(v_{d_1}, u_d)$ = intensidad resultante de la primera fila
- $I_{pix}(v_{d_2}, u_d)$ = intensidad resultante de la segunda fila

Estas operaciones se implementan en el código de MATLAB para la obtención de la imagen sin distorsión a partir de la imagen de partida (distorsionada).

3. IMAGEN 1. BICONVEXA

Para esta imagen, se ha empleado la lente que tiene como coeficiente de distorsión radial $k_{r_1} = -0.4370$, ya que se está trabajando sobre una imagen biconvexa. El resultado obtenido es el siguiente:

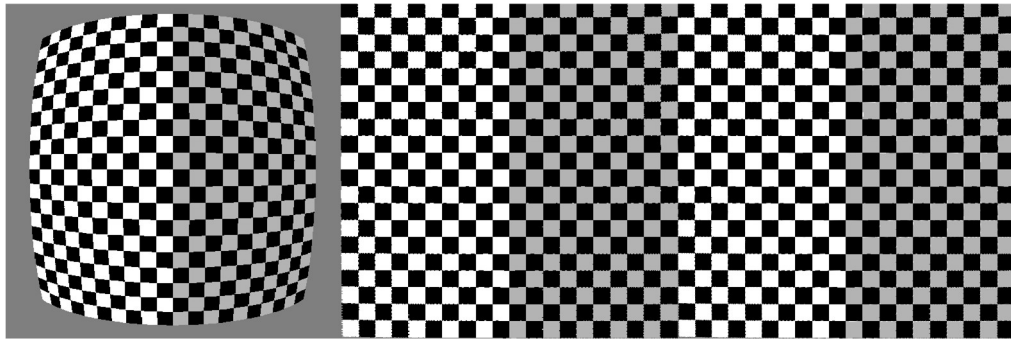


Ilustración 6. Resultados de la imagen 1: Imagen de partida – Nearest neighbour – Bilineal

A modo de ver las diferencias entre la imagen resultante aplicando un interpolador por el criterio del vecino más cercano y empleando el interpolador bilineal, se va a hacer una resta de las imágenes en valor absoluto para ver que no se obtiene el mismo resultado, a pesar que aparentemente lo sean.

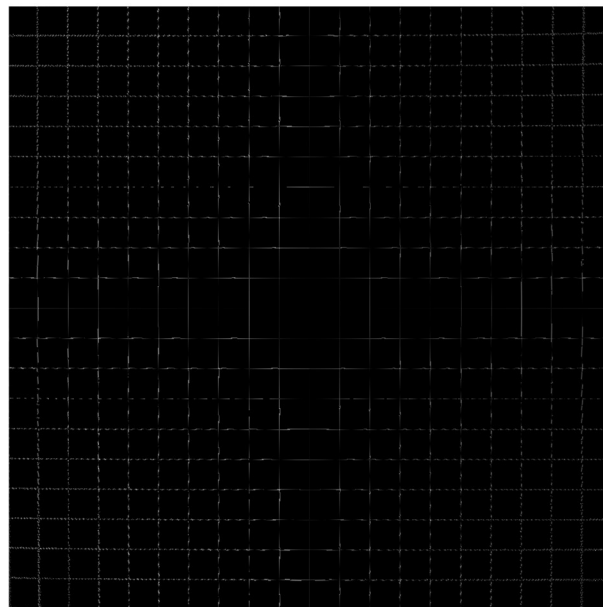


Ilustración 7. Diferencias entre interpolador de vecino más cercano y bilineal de la imagen 1

4. IMAGEN 2. CÓNCAVA

A diferencia de la imagen anterior, en este caso se ha empleado la lente que tiene como coeficiente de distorsión radial $k_{r_1} = +0.4370$, ya que se está trabajando sobre una imagen cóncava. El resultado obtenido es el siguiente:

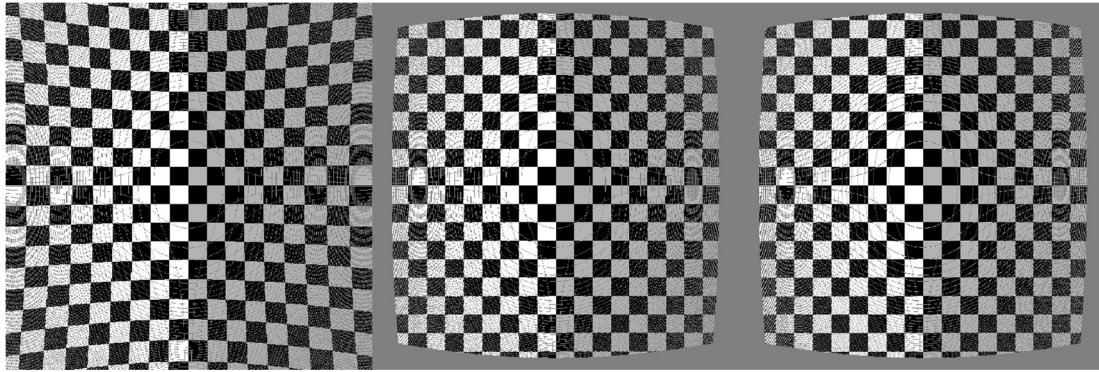


Ilustración 8. Resultados de la imagen 2: Imagen de partida – Nearest neighbour – Bilineal

Haciendo la comparación entre el resultado del vecino más cercano y el bilineal:

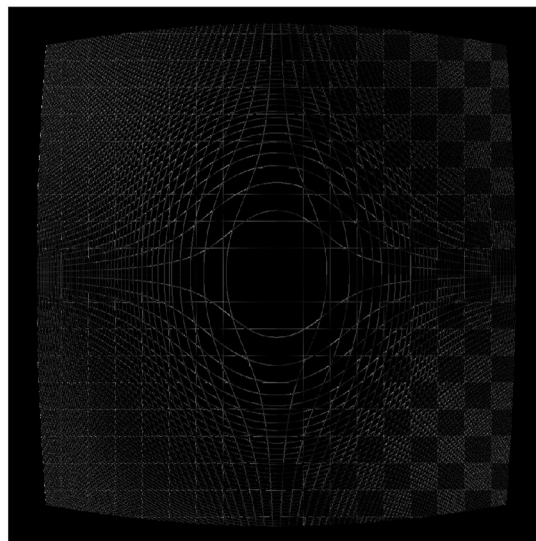


Ilustración 9. Diferencias entre interpolador de vecino más cercano y bilineal de la imagen 2

5. CÓDIGO IMPLEMENTADO EN MATLAB

A modo de hacer el código más claro, se ha separado el procedimiento en dos scripts distintos: uno llamado *ej_pract2.m* en el que está el código principal del ejercicio y es el código a ejecutar; y otro llamado *corrige.m* en el que está el procedimiento para obtener la imagen resultante corregida (sin distorsión), tanto con el interpolador del vecino más cercano como con el interpolador bilineal.

5.1 SCRIPT EJ_PRACT2.M

```
% -----  
%  
% Ejercicio práctico 2 - Eliminación de distorsión en imagen  
% -----  
%  
% ----- Parámetros de la cámara -----  
f = 4.2e-3; %Distancia focal  
  
N = 1000;  
M = 1000;  
  
u0 = N/2 + 1;  
v0 = M/2 - 2;  
  
% Sensor de la cámara  
h = 4.96e-3;  
w = 3.52e-3;  
  
% K de distorsión radial:  
% Lente 1:  
% kr1 = -0.4320;  
  
% Lente 2  
% kr1 = 0.4320;  
  
% Dimensiones efectivas del píxel:  
rho_x = w/N;  
rho_y = h/M;  
  
% Longitudes focales efectivas:  
fx = f/rho_x;  
fy = f/rho_y;  
  
% -----  
%  
% Leemos la imagen distorsionada  
% Imagen 1  
f1 = double(imread('chessBoardDistorted1.jpg'));  
  
% Imagen 2  
f2 = double(imread('chessBoardDistorted2.jpg'));
```



```

% Se inicializan las matrices con color gris tanto para el interpolador
del vecino más cercano como para el bilineal
vecino = double(128*ones([N,M]));
bilineal = double(128*ones([N,M]));

% Correccion de imagen 1
% function [vecino, bilineal] = corrige(imagen_distorsionada)
kr1 = -0.4320;
imagen_distorsionada = f1;
corrige;
fv1 = vecino;
fb1 = bilineal;

vecino = double(128*ones([N,M]));
bilineal = double(128*ones([N,M]));

% Correccion de imagen 2
kr1 = 0.4320;
imagen_distorsionada = f2;
%[fv2, fb2] = corrige(f2);
corrige;
fv2 = vecino;
fb2 = bilineal;

figure(1);
imshow(uint8([f1,fv1,fb1]));
figure(2);
imshow(uint8(abs(fv1-fb1)));

figure(3);
imshow(uint8([f2,fv2,fb2]));
figure(4);
imshow(uint8(abs(fv2-fb2)));

```

5.2 SCRIPT CORRIGE.M

```

% Transformacion de coordenadas sin a con distorsion pixel a pixel
for u = 0:N-1
    for v = 0:M-1
        % Cálculo de coordenadas normalizadas
        xn = (u-u0)/fx;
        yn = (v-v0)/fy;

        % u = xn*fx + u0;
        % v = yn*fy + v0;

        % Cálculo de la r2
        r2 = xn^2 + yn^2;

        % Coordenadas distorsionadas:
        xn_d = xn*(1+kr1*r2);
        yn_d = yn*(1+kr1*r2);
    end
end

```

```

% Coordinadas del píxel distorsionadas:
u_d = xn_d*fx + u0;
v_d = yn_d*fy + v0;

%Si estamos dentro de la imagen, calculamos intensidades de
los píxeles. Si no, se mantiene en gris
if(u_d >= 0 && u_d <= N-1 && v_d >= 0 && v_d <= M-1)
    % Criterio de interpolación al vecino más cercano
    vecino(u+1,v+1) =
imagen_distorsionada(round(u_d)+1,round(v_d)+1);

    % Criterio de interpolacion bilineal:

    % Identificamos los dos pixeles mas proximos a u_d. Estos
seran el
    % pixel que se obtenga de la transformacion y el que
indique el
    % redondeo (izquierda o derecha)

    % Direccion u
    u_d_i = floor(u_d); %u_d_i --> pixel actual
    if(floor(u_d) == round(u_d))
        u_d_j = u_d_i - 1; %u_d_j --> si está por debajo de
0.5
    else
        u_d_j = u_d_i + 1; %u_d_j --> si está por encima de
0.5
    end

    % u_d_1 --> píxel situado a la izquierda
    % u_d_2 --> píxel situado a la derecha
    if(u_d_i > u_d_j)
        u_d_1 = u_d_j;
        u_d_2 = u_d_i;
    else
        u_d_1 = u_d_i;
        u_d_2 = u_d_j;
    end

    % Si alguno de los píxeles están en el borde, se
considera
    % el mismo
    if(u_d_j < 0 || u_d_j >= N)
        u_d_1 = u_d_i;
        u_d_2 = u_d_i;
    end

    % Direccion v
    v_d_i = floor(v_d); %v_d_i --> pixel actual
    if(floor(v_d) == round(v_d))
        v_d_j = v_d_i - 1; %v_d_j --> si está por debajo de
0.5
    else
        v_d_j = v_d_i + 1; %v_d_j --> si está por encima de
0.5
    end

```

```

        % v_d_1 --> píxel situado encima
        % v_d_2 --> píxel situado debajo
        if(v_d_i > v_d_j)
            v_d_1 = v_d_j;
            v_d_2 = v_d_i;
        else
            v_d_1 = v_d_i;
            v_d_2 = v_d_j;
        end

        % Si alguno de los píxeles están en el borde, se
considera
        % el mismo
        if(v_d_j < 0 || v_d_j >= M)
            v_d_1 = v_d_i;
            v_d_2 = v_d_i;
        end

        % Ponderamos los valores de intensidad segun proximidad
en u.
        I_ud_v1 = (u_d_2 - u_d) * imagen_distorsionada(u_d_1 + 1,
v_d_1 + 1) + ...
            (u_d - u_d_1) * imagen_distorsionada(u_d_2 + 1, v_d_1
+ 1);

        I_ud_v2 = (u_d_2 - u_d) * imagen_distorsionada(u_d_1 + 1,
v_d_2 + 1) + ...
            (u_d - u_d_1) * imagen_distorsionada(u_d_2 + 1, v_d_2
+ 1);

        % Si nos encontramos en un borde vertical
        if(u_d_j < 0 || u_d_j >= N)
            I_ud_v1 = imagen_distorsionada(u_d_i + 1, v_d_1 + 1);
            I_ud_v2 = imagen_distorsionada(u_d_i + 1, v_d_2 + 1);
        end

        % Ponderamos los valores de intensidad segun proximidad
en v.
        bilineal(u+1,v+1) = (v_d_2 - v_d) * I_ud_v1 + ...
            (v_d - v_d_1) * I_ud_v2;

        % Si nos encontramos en un borde horizontal
        if(v_d_j < 0 || v_d_j >= M)
            bilineal(u+1,v+1) =
imagen_distorsionada(u+1,v_d_i+1);
        end
    end
end
end

```

6. CONCLUSIONES

En este ejercicio se han trabajado con imágenes bastante simples, similar a un tablero de ajedrez (contornos simples, líneas totalmente horizontales o verticales e intensidades B/W), por lo que al usar un método de interpolación u otro no se aprecia diferencia a simple vista a no ser que se resten las imágenes obtenidas y se vean reflejadas en otra figura las diferencias que hay entre los resultados obtenidos por cada interpolador (Ilustraciones 7 y 9). Si en vez de haber trabajado con una imagen con geometrías simples se hubiera trabajado con una imagen con geometrías más complejas, diferentes intensidades y que los contornos no formen ángulos horizontales o verticales por completo, en principio se apreciaría una diferencia mayor al usar un interpolador diferente. Sin embargo, en este caso, parece no importar a priori qué interpolador usar, ya que se obtienen buenos resultados en ambos casos.