

Universidad
Rey Juan Carlos

Fundamentos Matemáticos

PRÁCTICA I :

Cálculo Variacional y Procesado de Imágenes

04/06/2022

Luis ROSARIO TREMOULET

I. Primera parte : Denoising

I.I Introducción

Esta primera parte introduce el problema del procesado de imágenes mediante cálculo variacional. En concreto intentaremos resolver un problema de eliminación de ruido sobre imágenes. Las imágenes elegidas para este problema se pueden ver en la figura 1.

Matemáticamente el denoising consiste en resolver un problema de minimización. Intentaremos invertir el degradado (ruido) generado en las imágenes de la figura 1. La resolución de este problema se intentará utilizando Matlab.

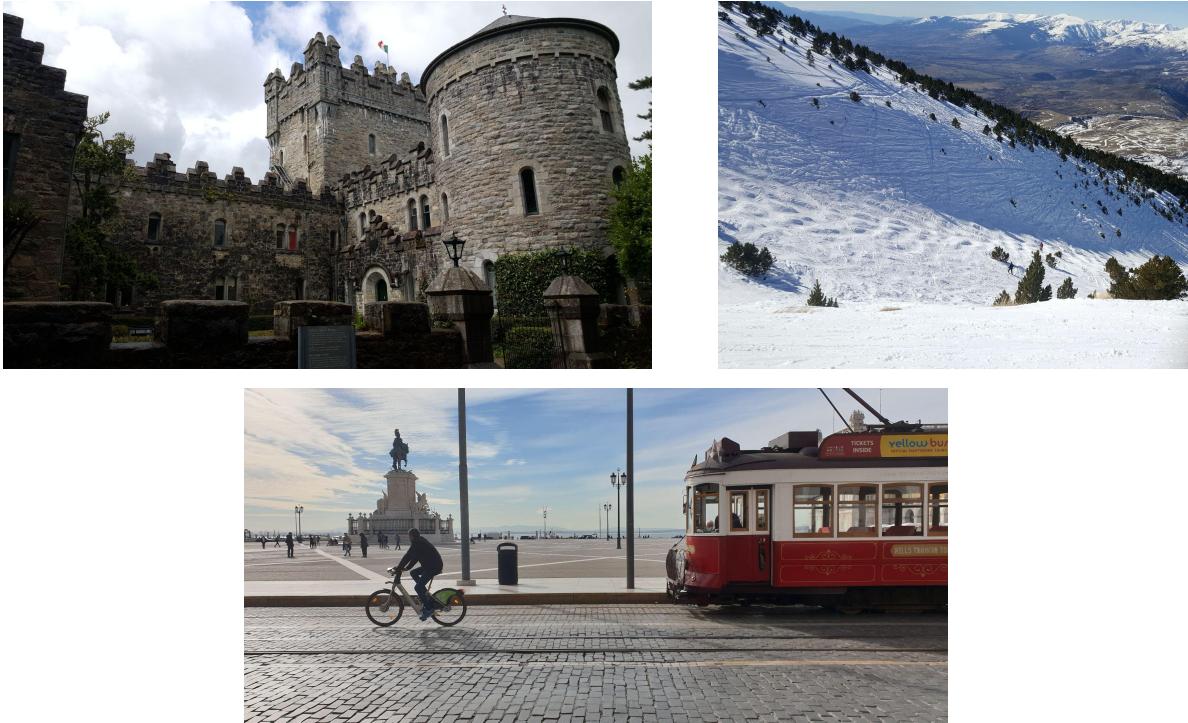


Figura 1: Imágenes utilizadas para la primera parte

I.II Generación del ruido : Noising

Primero necesitaremos generar una imagen con ruido, esto nos permitirá realizar nuestro "Denoising". Esta degradación se traducirá por el modelo :

$$f = u + \eta$$

donde u es la imagen ideal, η es un ruido que consideraremos y f son los datos que tenemos y queremos mejorar.

También asumimos que el ruido será aditivo, blanco y Gaussiano.

Para generar ese ruido AWGN (Additive White Gaussian Noise) en matlab :

```
%% Deterioramos la imagen
small_noise = 0.05;
medium_noise = 0.15;
big_noise = 0.3;

Noise_STD = big_noise;|
noise      = Noise_STD*randn(size(im_final));
im.blur    = im_final + noise;
```

Noise_STD = la desviación típica del ruido (aquí se puede elegir tres intensidades de ruido)

noise = η

im_final = u

im.blur = f

Para estimar el ruido en las imágenes utilizaremos el script **estimate_noise.m** donde se pueden apreciar unos posibles resultados en la tabla 1.

	No noise	Small	Medium	Big
Valor de la desviación típica del ruido	0	0.05	0.15	0.3
Resultado de estimate_noise.m	1.770	8.144	21.255	37.868

Tabla 1: Resultados del estimate_noise.m con la imagen 1 (Castillo) con diferentes valores de desviación típica

El objetivo de la práctica será obtener una imagen con menos ruido, es decir minimizar el valor obtenido por el script `estimate_noise.m`.

I.III Resolución del problema : Descenso de Gradiente

Sabiendo que el problema de denoising es un problema de minimización, utilizaremos el descenso de gradiente para solucionarlo. El descenso de gradiente consiste en calcular el gradiente de una función que queremos minimizar, y “moverse” en la dirección de dicho gradiente en sentido contrario (descenso). Concretamente tenemos :

$$\min_u E(u) = \min_u \left\{ \lambda \int_{\Omega} (u - f)^2 dx + \int_{\Omega} |\nabla u|^2 dx \right\}$$

donde se tiene que minimizar dos términos :

1. La energía del módulo del gradiente (a priori)
2. La energía del residuo

Para resolver este problema de descenso de gradiente de manera numérica, discretizamos la ecuación anterior utilizando diferencias finitas. Esto nos daria la formula iterativa :

$$u_{k+1} = u_k - \Delta t (-\Delta u_k + \lambda(u_k - f))$$

donde la solución deseada se encuentra en el infinito. Como no es posible iterar infinitamente, tendremos que elegir un parámetro para detener el cálculo después de n iteraciones (lo llamaremos Nit).

I.IV Denoising linear diffusion

Para completar el archivo *Denoising_Linear_Diffusion.m* basta con aplicar la fórmula anterior del descenso de gradiente (manera numérica) :

```
% Algoritmo
ux = gradx(u);
uy = grady(u);
% Laplaciano
lap = div(ux,uy);
wk = -lap + lambda*(u-f); % completar
% Descenso
u = u - dt * wk; % completar
```

$$u_{k+1} = u_k - \Delta t (-\Delta u_k + \lambda(u_k - f))$$

dt = Δt (tamaño del paso)

wk = gradiente del funcional de energía

u = es el resultado (que cambia a “cada” iteración)

I.IV Energy

El archivo *Energy.m* sirve para calcular las energías: Total, a priori y de fidelidad. Gracias a la primera fórmula podemos completar el fichero Matlab :

```
ux = gradx(u);
uy = grady(u);

% Prior
dim = size(u);
dim_total = dim(1)*dim(2);

m_gradient = sqrt(ux.^2 + uy.^2);
prior = sum(m_gradient(:).^2) / dim_total * 0.5;

fi = (u-f).^2;
fidelity = sum(lambda(:).*fi(:) / dim_total) * 0.5;

% Total Energy
energy= prior + fidelity;
```

$$\min_u E(u) = \min_u \left\{ \lambda \int_{\Omega} (u - f)^2 dx + \int_{\Omega} |\nabla u|^2 dx \right\}$$

donde **fidelity** es la fidelidad, término que hay que minimizar (energía del residuo). **Dim_total** es el número total de píxeles en la imagen. **Prior** es el a priori, la energía del descenso de Gradiente (o semilla en el caso de la primera iteración). **Energy** es la energía total (la suma del a priori y de la fidelidad).

I.V Los parámetros

Tenemos más o menos una docena de parámetros configurables cuando se trata de ejecutar el algoritmo de eliminación de ruido.

```
%% Selección de parámetros
close all
% Proporcionamos los parámetros para nuestro algoritmo
varin.lambda = 0.39; % hyperparámetro de fidelidad
varin.Nit = 82; % número de iteraciones del algoritmo
varin.dt = 1e-2; % tamaño del paso
varin.f = im.blur; % imagen ruidosa
varin.Verbose = 1; % Verbose
varin.im_org = im_final; % Imagen original para el cómputo de la PSNR
% Los parámetros se agrupan en un struct por simplicidad
```

Primero tenemos los parámetros básicos” : ***im_org*** que es la imagen original (necesitada para el cómputo de la PSNR). La PSNR es la métrica que nos permitirá valorar la calidad de la imagen (Pick to Signal Noise Ratio), mayor es, mejor es la reducción de ruido.

El parámetro ***f*** que es la imagen ruidosa.

Verbose es el parámetro que nos permite ver más o menos detalles sobre el resultado como gráficas.

En segundo lugar tenemos los parámetros que tienen una repercusión sobre la resolución del denoising : el parámetro ***u*** es la imagen semilla, normalmente se suele usar una imagen parecida al resultado final (aquí será la imagen ruidosa).

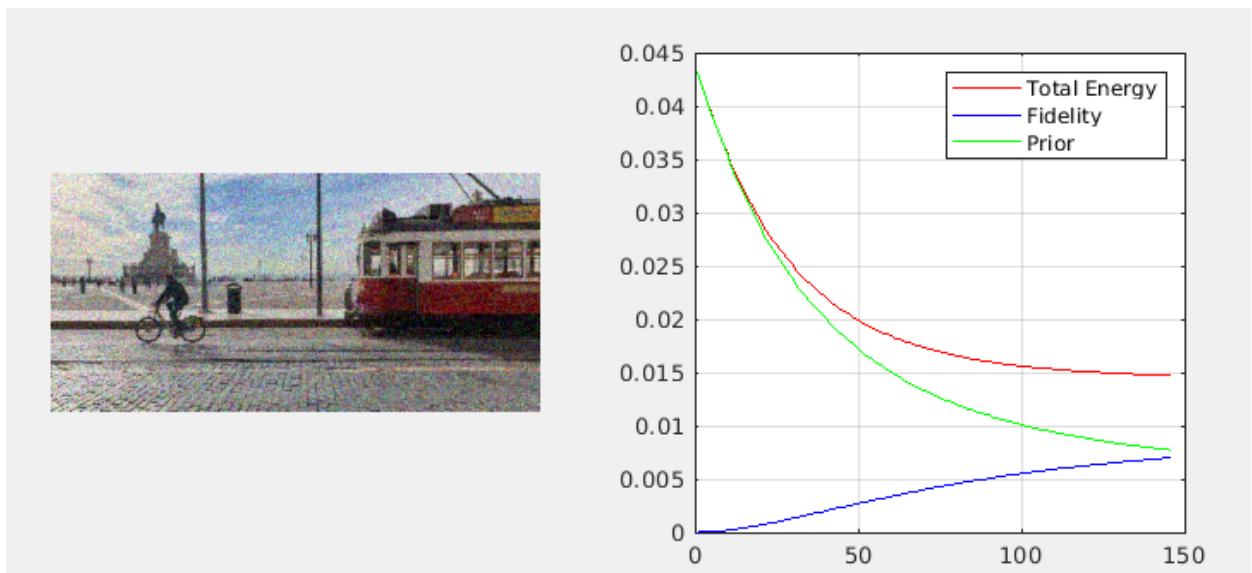


Figura 2: Resultado con la semilla igual a la imagen ruidosa

Como se puede ver en la figura 2, usar la imagen ruidosa provoca una aumentación y no una minimización de la fidelidad (aunque minimicemos la energía total). Esto es porque paso a paso estamos minimizando el ruido (nos alejamos de la semilla).

También tenemos el parámetro ***lambda*** (λ) es el hyper parámetro de fidelidad. Más alto será el lambda más fiable será el resultado a la semilla, para que el algoritmo haga una reducción de ruido este parámetro tiene que ser superior a 0.

El parámetro dt es el tamaño del paso para el descenso de gradiente.

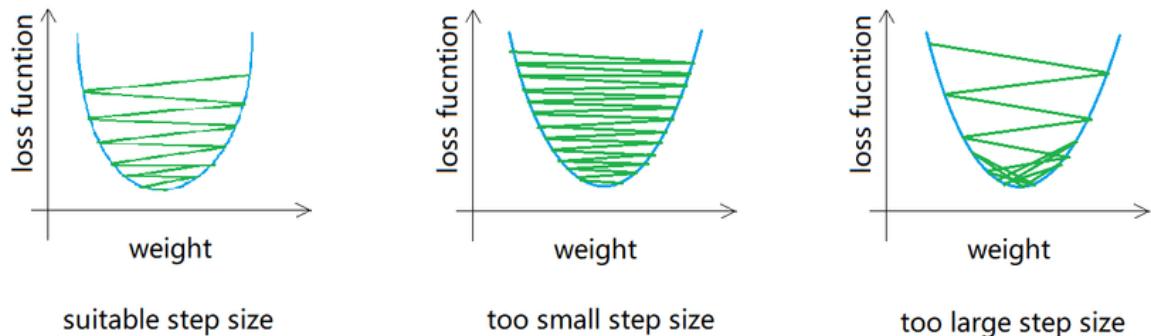


Figura 3: Diferentes valores de dt para descenso de gradiente

Como se puede observar en la figura 3, cuanto más alto sea este parámetro, más rápido minimizará el algoritmo (en menos iteraciones), pero más probabilidad tiene de nunca encontrar el mínimo.

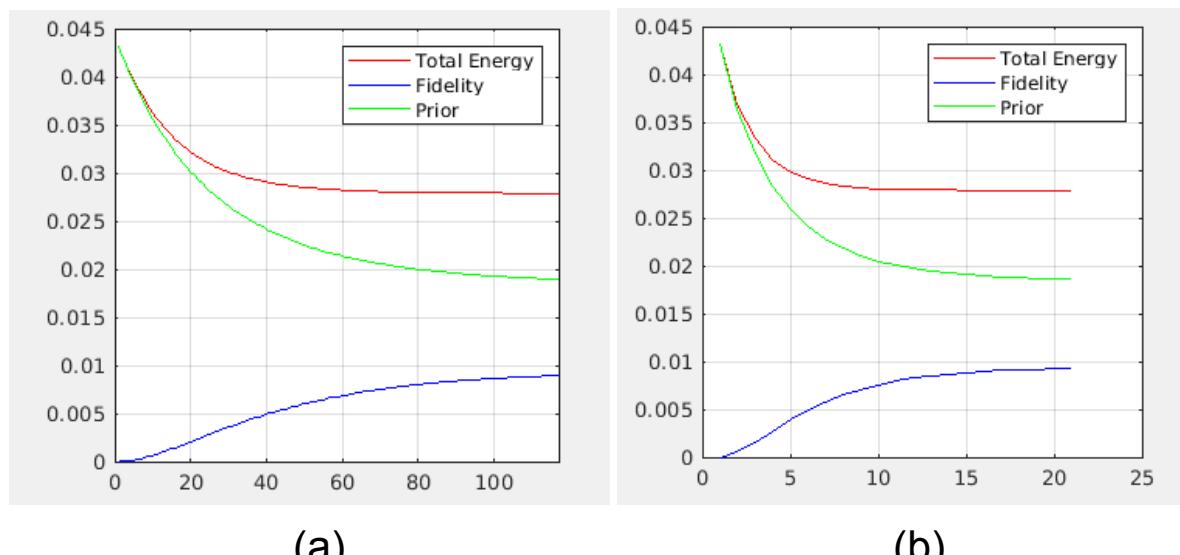


Figura 4: Grafica comparando el parámetro dt :
(a) $dt = 1e-2$ (b) $dt = 7e-2$

Como se puede ver en la figura 4 el algoritmo minimiza en menos iteraciones cuando el parámetro dt es mayor.

Finalmente el último parámetro configurable es el número de iteraciones : **Nit**. Las iteraciones definen el número máximo de iteraciones para el descenso de gradiente. Su valor se puede encontrar mirando la métrica PSNR (cuando la PSNR parece ser estable no se necesitan más iteraciones).

En la figura 5 se puede ver que no hace faltan más de 37 iteraciones.

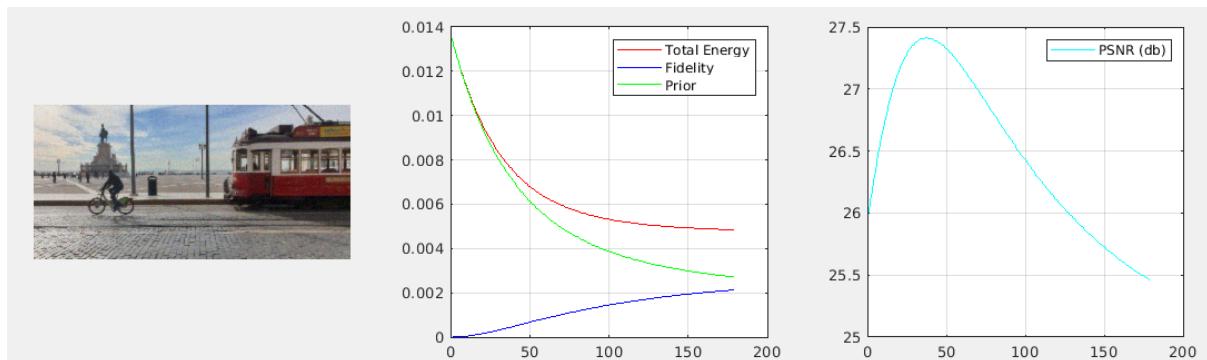


Figura 5: Resultado de un Denoising con 160 iteraciones
y un $dt = 1e-2$

I.VI La imagen de origen

El tiempo de ejecución de cada iteración no solo depende del parámetro dt , también depende del tamaño de la imagen (número total de píxeles), más números de píxeles hay en la imagen y más tiempo de cómputo es necesario. El número de canales de la imagen también influye en el tiempo de cómputo, por ejemplo una imagen en rgb (3 canales) es más compleja que una imagen en rango de grises (1 canal).

I.VII Resultados y conclusión

Para los resultados he utilizado dos imágenes de tamaño diferente y con diferentes desviaciones del ruido. La métrica que utilizaré para comparar los resultados será la PSNR como se ha visto previamente. Como no se podrá bien apreciar los resultados, el código utilizado será disponible y los parámetros utilizados serán comentados.



Figura 6: Resultado final del Denoising 1

En la Figura 6 hemos utilizado una deterioración con un valor de 0.05 (desviación típica del ruido), un lambda de 0.56, 36 iteraciones y un paso de tamaño 1e-2. El resultado es bastante bueno, se nota una reducción del ruido con solo ver la imagen, y se obtiene un valor de la PSNR = 27.4105.

Para el segundo ejemplo utilizaremos una deterioración con un valor más alto (0.3). Los parámetros ideales encontrados son un lambda de 0.4, 249 iteraciones y un paso de tamaño 4e-2 (como la imagen contiene muchos más pixeles que la primera el objetivo era minimizar más rápidamente). Como la imagen contiene mucha más deterioración (comparado con el primer ejemplo), el resultado obtiene una PSNR más baja, pero pasa de 10.4785 a 18.487. Por lo cual el resultado es muy bueno como se puede apreciar en la figura 7 y 8.

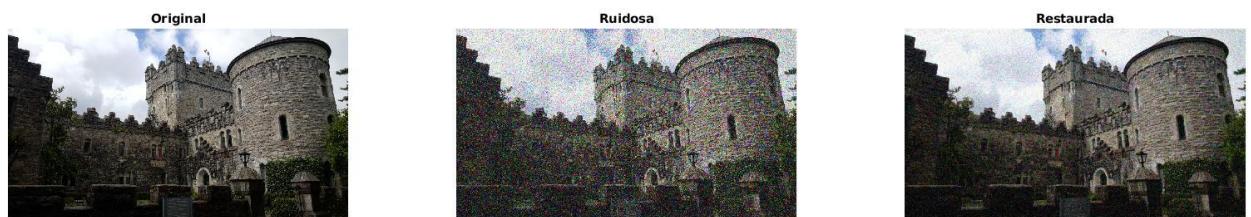


Figura 7: Resultado final del Denoising 2

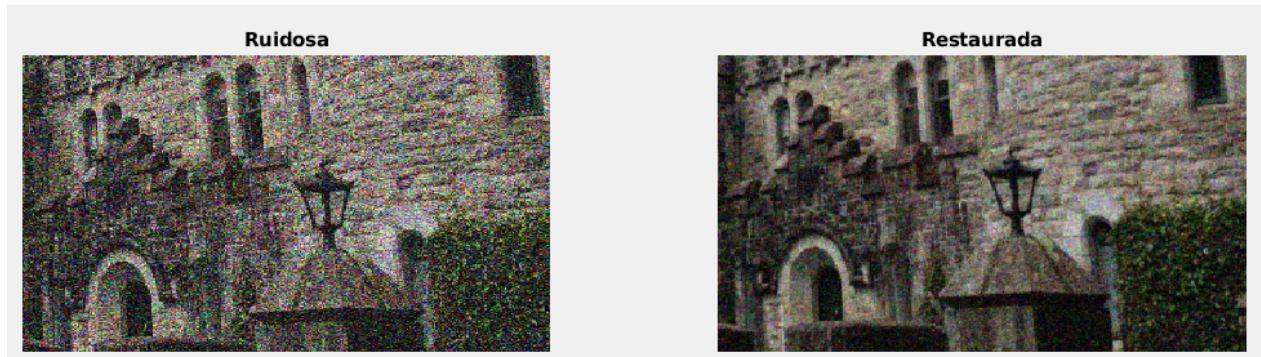
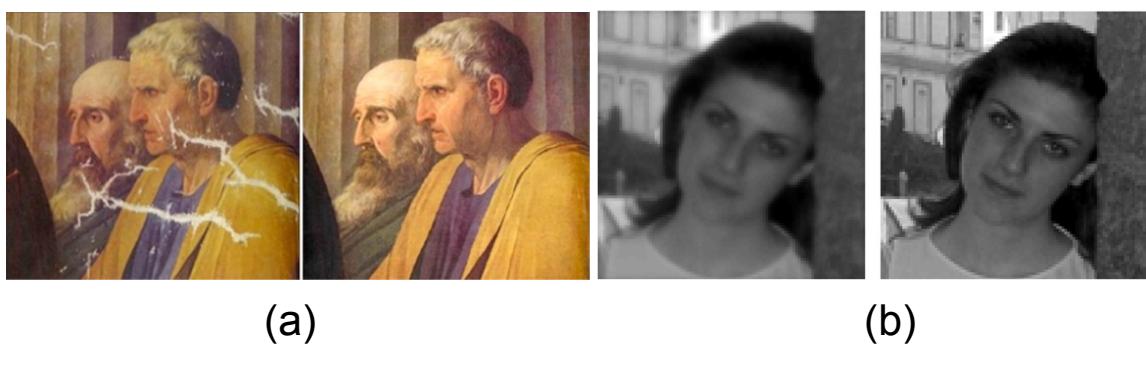


Figura 8: Zoom del resultado final del Denoising 1

II. Segunda parte : Impaiting y Deconvolución

II.I Introducción

En esta segunda y última parte intentaremos resolver el problema de impaiting y deconvolución en imágenes. El inpainting es una técnica de reconstrucción de imágenes dañadas que consiste en recuperar regiones que, o bien se han perdido, o están tan deterioradas que han de ser completamente sustituidas. La deconvolución es la reconstrucción de una imagen que ha sido convolucionada con un filtro k .



*Figura 9: Ejemplos de impaiting y deconvolución:
(a) impaiting (b) deconvolution*

II.II El parámetro p

Los parámetros configurables son más o menos los mismos que los de la primera parte. Sin embargo, se añade un parámetro, la p . Generalmente este parámetro tiene un valor de $1 \leq p < \infty$ sabiendo que el rango típicamente utilizado en procesamiento de imágenes es $1 \leq p \leq 2$. Cuanto más pequeño sea este parámetro más agresivo será el proceso de difusión. Intentaremos comparar los resultados con diferentes valores de p .

II.V Deconvolución : Resolución en Matlab

Como estamos usando una función que contempla distintos valores de p , calculamos el descenso de gradiente con la ecuación de Euler-Lagrange de manera numérica (visto en clase) :

```
% Algoritmo
ux = gradx(u, 'forward');
uy = grady(u, 'forward');

b = sqrt(ux.^2 + uy.^2 + epsilon.^2).^(p-2);
% Laplaciano
lap = div(b.*ux, b.*uy);
% Descenso

u = u - dt*(-lap + Lambda*RT(R(u)-f));
```

$$u_{k+1} = u_k - \Delta t (-\operatorname{div}(|\nabla u_k|_\epsilon^{p-2} \nabla u_k) + \lambda(\mathbf{x}) k^T * (k * u_k - f))$$

donde R y RT son dos funciones que aplican el kernel k o k transpuesta.

II.VI Deconvolución : Resultados

Para esta parte utilizaremos una imagen en tonos de grises y una en rgb (misma imagen pero con un dominio diferente).

Las dos imágenes han sido dañadas con un ruido de **0.1**.

El resultado para la imagen en blanco y negro se puede apreciar en la figura 10.

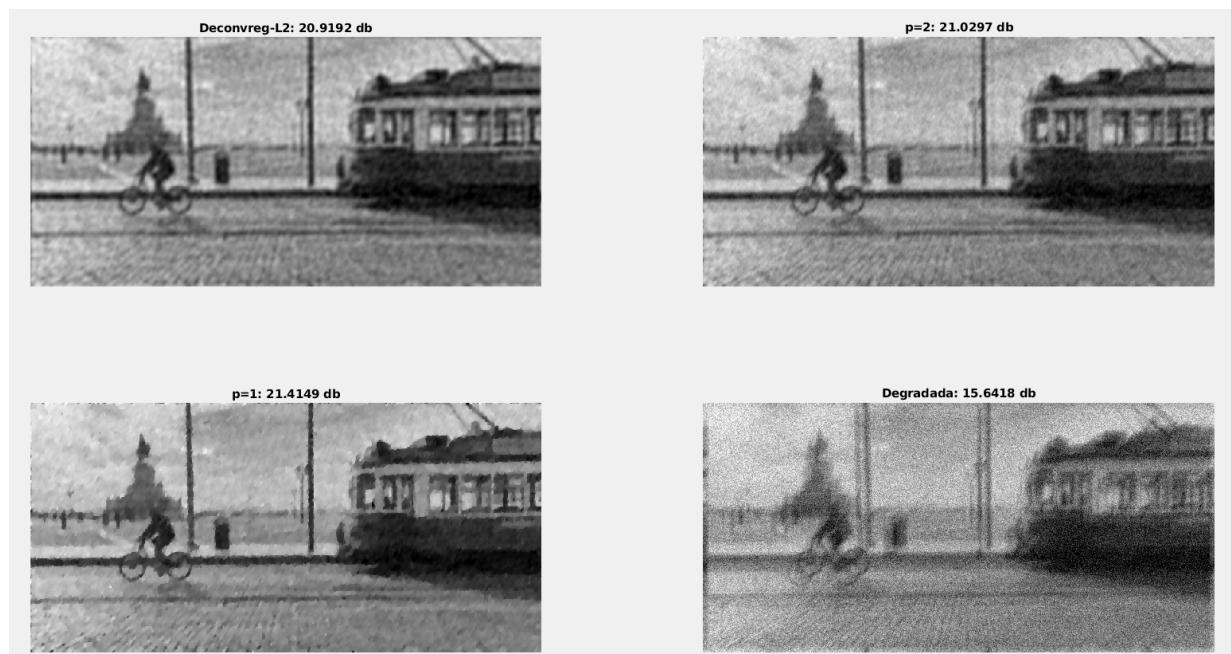


Figura 10: Resultado 1 de una deconvolución con ruido = 0.1

En la figura 10 podemos ver que el mejor resultado (con la PSNR más alta) es el resultado con el parámetro $p = 1$ (21.4149). Aunque el resultado con el parámetro $p = 1$ es el que tiene la PSNR más alta, el resultado se ve borroso (se parece a una pintura). En efecto, el resultado con $p = 2$ parece más nítido (mantiene las líneas rectas) y obtiene una PSNR casi idéntica a la de $p = 1$.

El resultado con el algoritmo propuesto por Matlab es muy parecido al filtrado no lineal ($p = 1$).

Los parámetros utilizados han sido : un paso de tamaño 5e-3, un lambda de 50 y 197 iteraciones para la $p = 1$ y un lambda de 2 y 615 iteraciones para la imagen con $p = 2$.

Usando los mismo parámetros obtenemos con la imagen en RGB esos resultados :

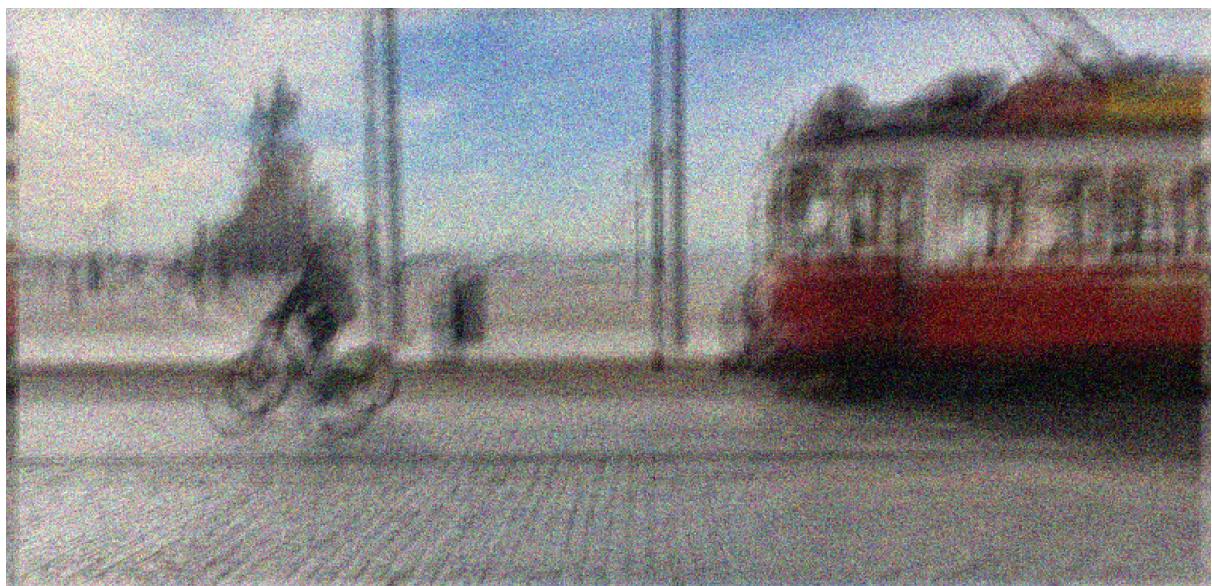


Figura 11: Imagen RGB ruidosa (0.1) para probar una deconvolución



Figura 12: Deconvolución de la Figura 11
con $p = 2$ (PSNR = 21.0373)



Figura 13: Deconvolución de la Figura 11
con $p = 1$ (PSNR = 21.4377)

Como los resultados de la imagen en blanco y negro, la figura 12 ($p = 2$) es menos nítida comparado con la figura 13 ($p = 1$) pero los bordes están menos afectados (ver las ruedas de la vici).

II.III Impaiting : Resolución en Matlab

El objetivo del impaiting es (usando el mismo funcional de energía) construir un lambda adaptativo, donde el lambda contiene los píxeles que “faltan” a la imagen.

```

% Algoritmo
ux = gradx(u, 'central');
uy = grady(u, 'central');

b = sqrt(ux.^2 + uy.^2 + epsilon.^2).^(p-2);
% Laplaciano
lap = gradx(b.*ux, 'central') + grady(b.*uy, 'central');
% Descenso

u = u - dt*(-lap + lambda.*(u-f));

```

La solución es la misma que en la deconvolución solo que el lambda es adaptativo (misma ecuación). Entonces, el lambda ya no es un escalar sino que es una “imagen”, por lo cual hay que multiplicarlo punto a punto.

II.III Impaiting : Resultados

Para esta parte intentaremos hacer denoising y también inpainting sobre la misma imagen (blanco y negro). La imagen seleccionada se puede observar en la figura 14, donde el objetivo es remover la estatua del caballo.



Figura 14: Imagen ruidosa (0.05) para el inpainting



Figura 15: Resultado del inpainting de la figura 14

Podemos ver, en los resultados con el parámetro $p = 1$ y 0.75 que el caballo ha completamente desaparecido, mientras que el resultado con $p = 2$ no se ha conseguido un impaiting perfecto (teniendo en cuenta que el lambda (mancha) es grande comparado con el tamaño de la imagen).

Para los tres resultados los parámetros utilizados son : un paso de $1e-2$ y un lambda de $100*\lambda$. Sin embargo no todos los resultados han necesitado las mismas iteraciones, 622 para $p = 1$, 500 para $p = 0.75$ y 855 para $p = 2$.