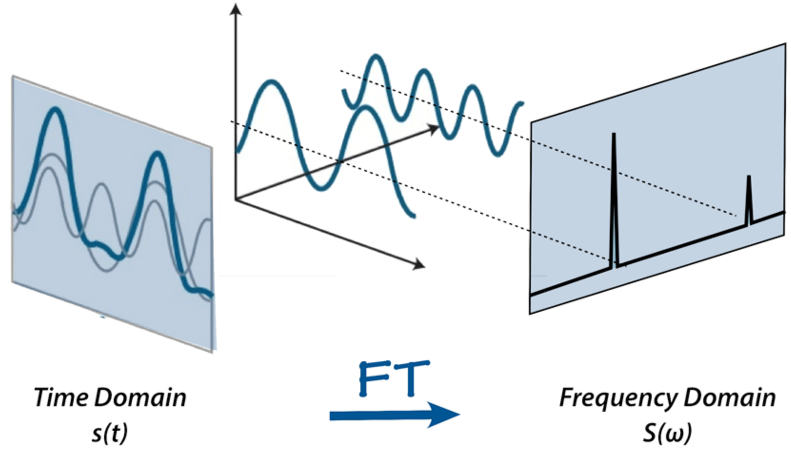


Visión Artificial

3. Transformación del dominio y espacial

Universidad Rey Juan Carlos
Escuela Técnica Superior Ingeniería de Telecomunicación

JOSÉ MIGUEL GUERRERO HERNÁNDEZ

EMAIL: JOSEMIGUEL.GUERRERO@URJC.ES

Índice de contenidos

1. Transformación del dominio (o frecuencia):
   - Transformada de Fourier
   - Transformada del coseno
   - Transformada de Wavelets
   - Ejemplos
2. Transformación espacial:
   - Transformación píxel a píxel
   - Transformaciones de vecindad
   - Transformaciones lógicas
   - Transformaciones geométricas
   - Ejemplos

Universidad Rey Juan Carlos | Escuela Técnica Superior Ingeniería de Telecomunicación

Índice de contenidos

1. Transformación del dominio (o frecuencia):
   - Transformada de Fourier
   - Transformada del coseno
   - Transformada de Wavelets
   - Ejemplos
2. Transformación espacial:
   - Transformación píxel a píxel
   - Transformaciones de vecindad
   - Transformaciones lógicas
   - Transformaciones geométricas
   - Ejemplos

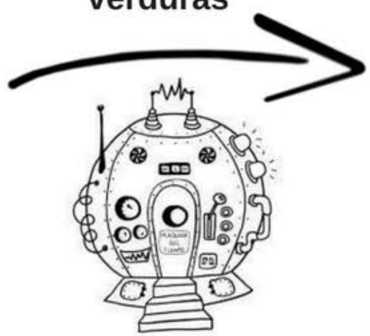
1. Transformación del dominio

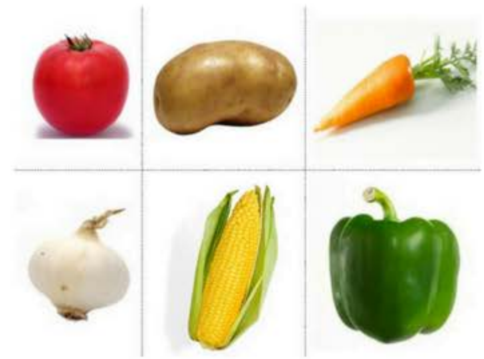
- Las imágenes normalmente se adquieren y se muestran en el **dominio espacial**, en el que los **píxeles adyacentes representan partes adyacentes** de la escena
- Sin embargo, las imágenes también se pueden adquirir en otros dominios, como el **dominio de frecuencia** en el que los **píxeles adyacentes representan componentes de frecuencia adyacentes**
- La visualización y el procesamiento de una imagen en dominios no espaciales pueden permitir la **identificación de entidades que pueden no detectarse tan fácilmente en el dominio espacial**

Índice de contenidos

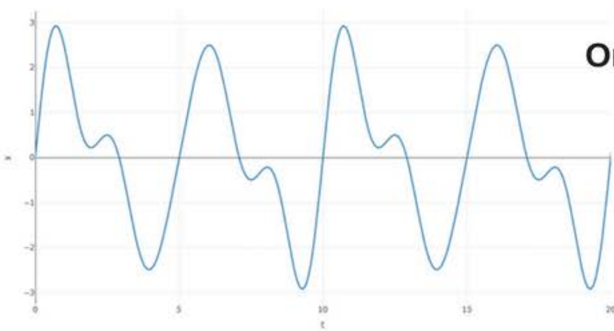
1.1. Transformada de Fourier

Separador de verduras

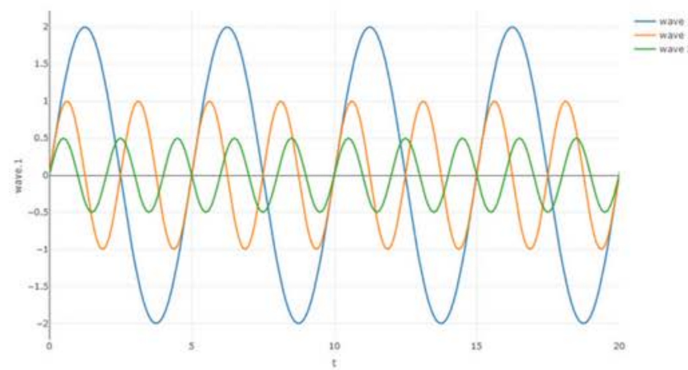




Separador de Ondas de Fourier



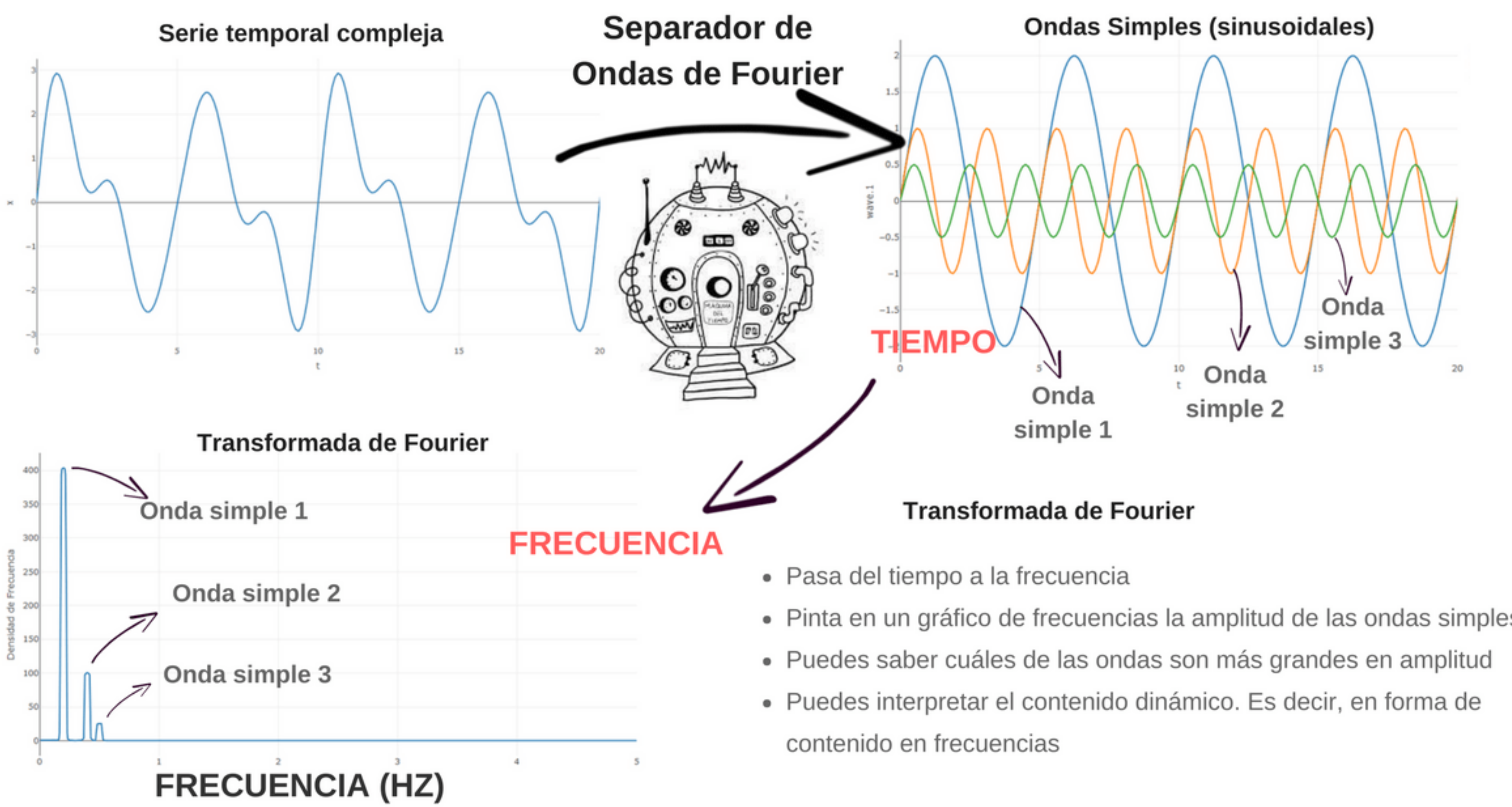
Serie temporal compleja



Ondas Simples (sinusoidales)

https://conceptosclaros.com/transformada-de-fourier/

1.1. Transformada de Fourier

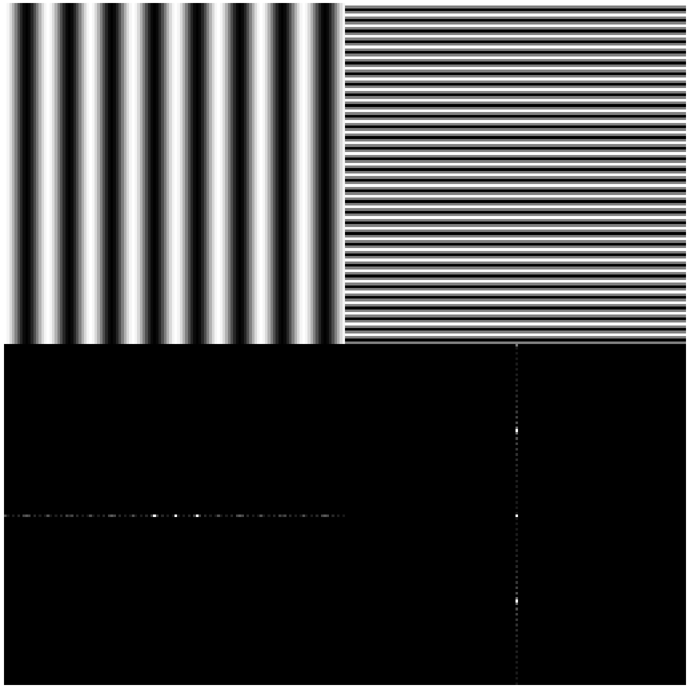


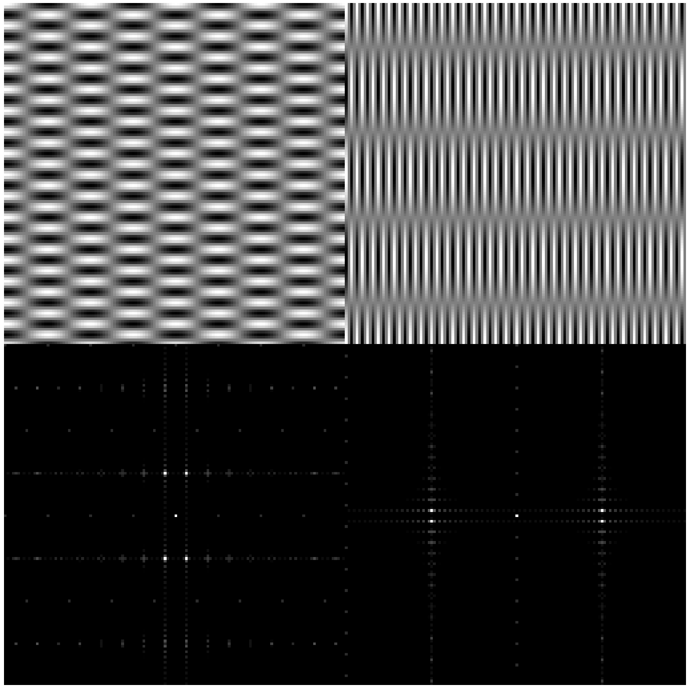
https://conceptosclaros.com/transformada-de-fourier/

1.1. Transformada de Fourier

- La transformación de Fourier es una representación de una imagen como una suma de exponenciales complejos de diferentes magnitudes, frecuencias y fases
- Desempeña un papel fundamental en una amplia gama de aplicaciones de procesamiento de imágenes, incluida la mejora, el análisis, la restauración y la compresión (coseno es el estándar JPEG)
- La transformada de Fourier muestra que una **imagen** puede ser construida por la **combinación de armónicos de frecuencias verticales y horizontales**
- A mayor frecuencia, más transiciones de la luminancia en menos píxeles en la dirección determinada por la componente

1.1. Transformada de Fourier





1.1. Transformada de Fourier

- Directa continua:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\cdot e^{-j2\pi(ux+vy)}dxdy$$

- Inversa continua:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\cdot e^{j2\pi(ux+vy)}dudv$$

- Directa discreta:

$$F(u,v) = \frac{1}{M\cdot N}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cdot e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

- Inversa discreta:

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\cdot e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

1.1. Transformada de Fourier

- La transformada de Fourier de una función real f(x) consta, en general, de una parte real *Re*(*u*,*v*) y otra imaginaria *Im*(*u*,*v*), de forma que:

$$F(u,v) = Re(u,v) + iIm(u,v)$$

- o también puede expresarse en términos de amplitud |*F*(*u*,*v*)| y fase ϕ(*u*,*v*):

$$|F(u,v)| = \sqrt{[Re(u,v)]^2 + [Im(u,v)]^2}$$

y

$$\phi(u,v) = tan^{-1}\left[\frac{Im(u,v)}{Re(u,v)}\right]$$

1.1. Transformada de Fourier

- Alta frecuencia espacial:

Señal unidimensional

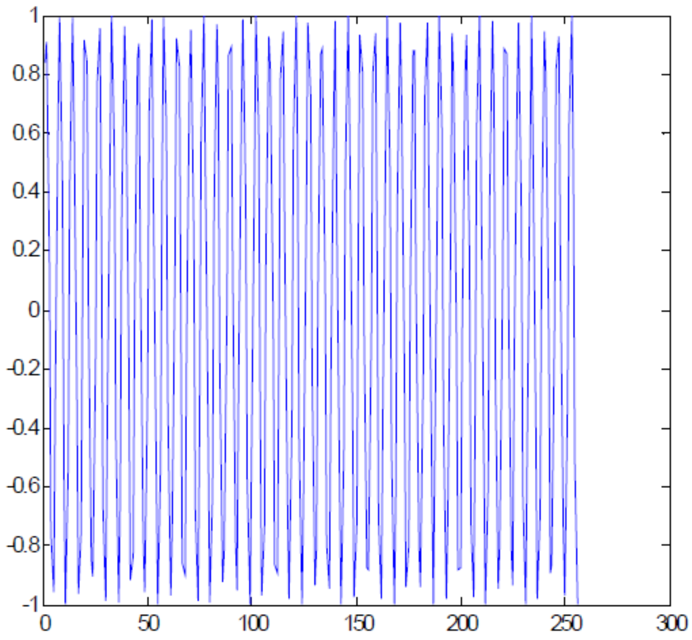
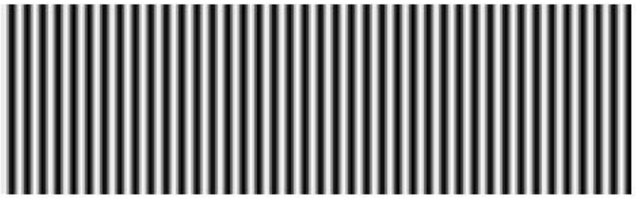
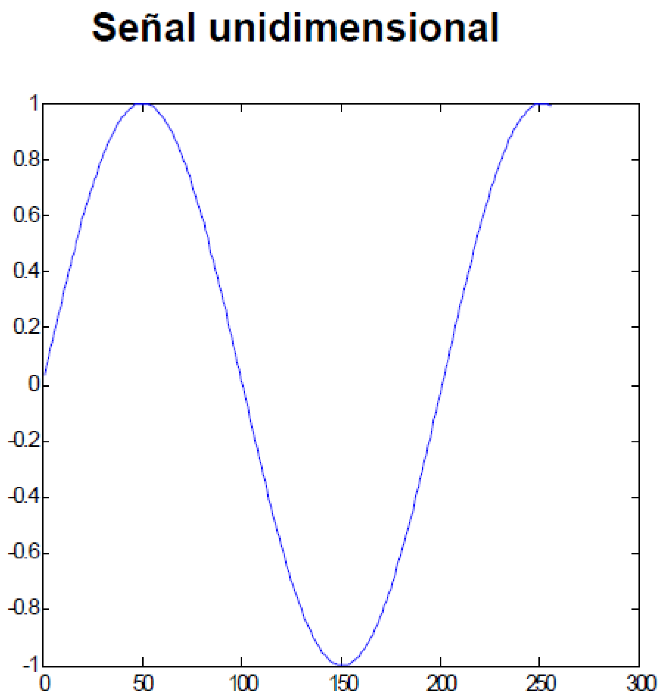


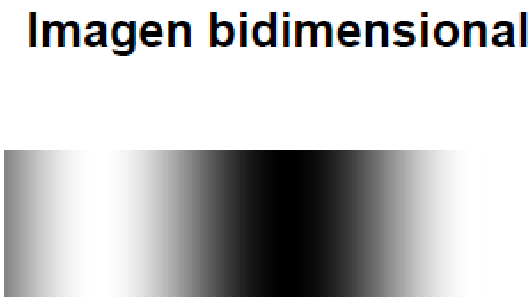
Imagen bidimensional



1.1. Transformada de Fourier

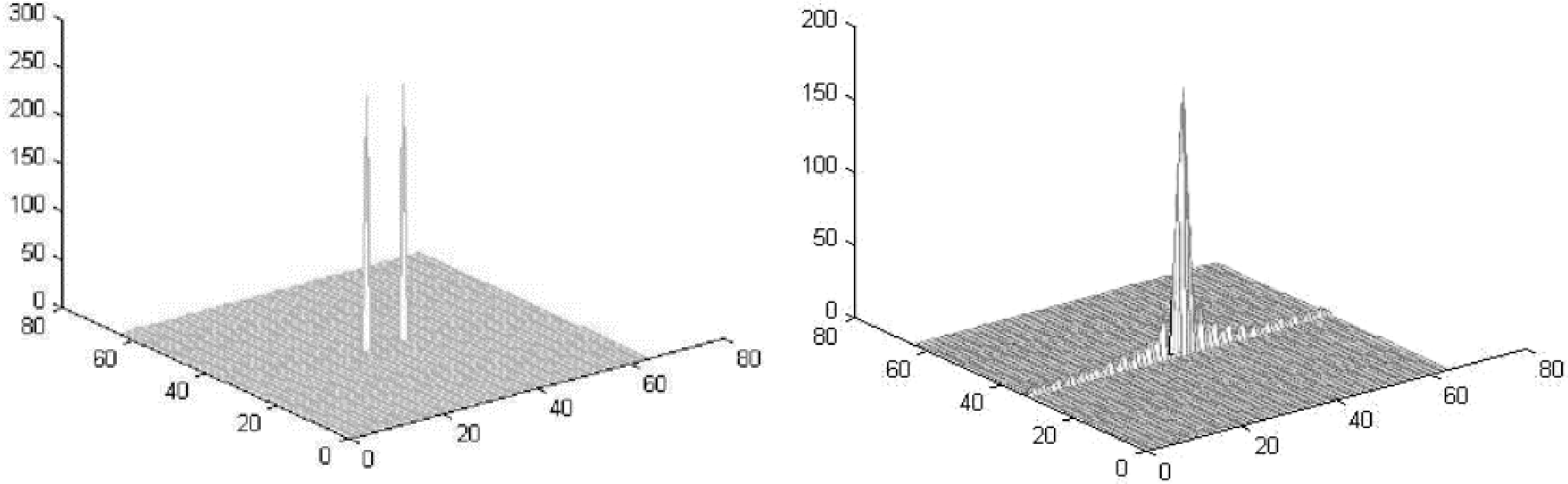
- Baja frecuencia espacial:





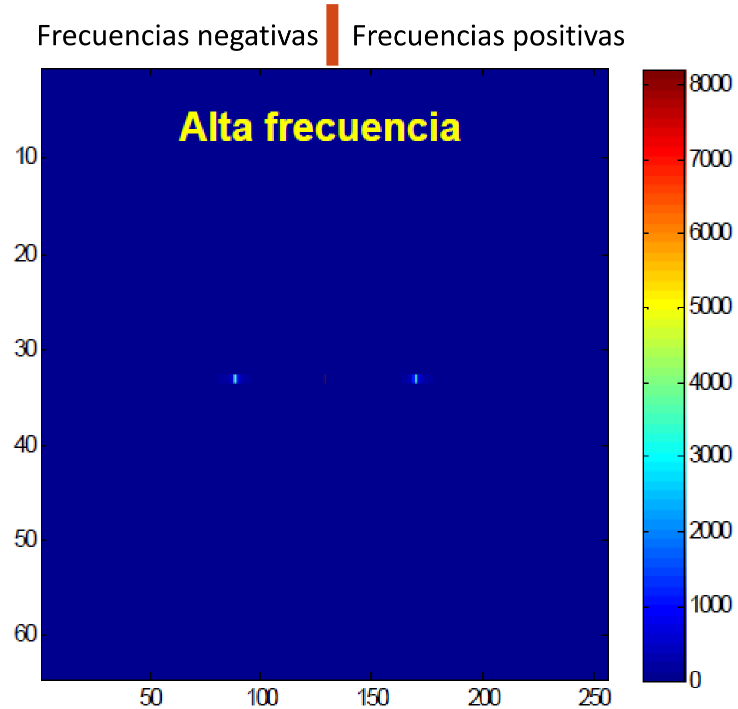
1.1. Transformada de Fourier

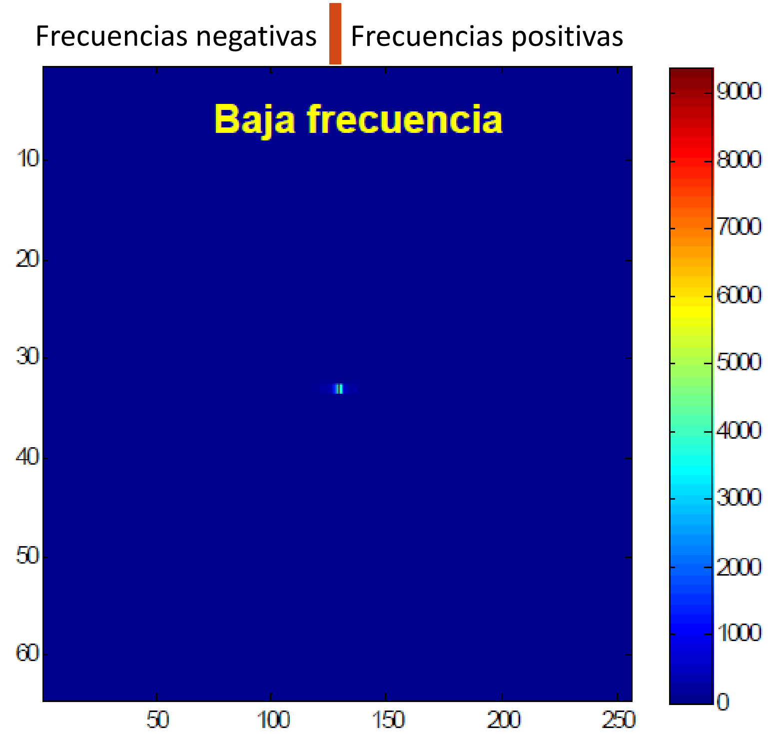
- Espectros de frecuencia, en formato manto:



1.1. Transformada de Fourier

- Espectros de frecuencia:





1.1. Transformada de Fourier

- Propiedades Transformada de Fourier Discreta bidimensional (TFD)

- Separabilidad:
  - Esta propiedad de la TFD esta relacionada con la posibilidad de calcular la TFD de una función bidimensional como una combinación de dos transformadas Fourier discretas, calculando primero una TFD sobre la variable de uno de los ejes y al resultado aplicarle de nuevo la TFD sobre la variable del otro eje
  - La ventaja que aporta esta propiedad es el hecho de poder obtener la transformada F(x,y) o la inversa f(x,y) en dos pasos, mediante la aplicación de la Transformada de Fourier 1-D o su inversa

- La linealidad:
  - La transformada de Fourier y su inversa son transformaciones lineales, es decir, poseen la propiedad distributiva respecto de la suma

1.1. Transformada de Fourier

- Propiedades Transformada de Fourier Discreta bidimensional (TFD)

- La traslación:
  - Tanto la transformada discreta de Fourier como la transformada inversa, son periódicas de periodo N

- La Simetría:
  - La transformada de Fourier de una función f(x,y) si es real, es simétrica conjugada. Esto provoca que:

$$|F(u,v)|=|F(-u,-v)|$$

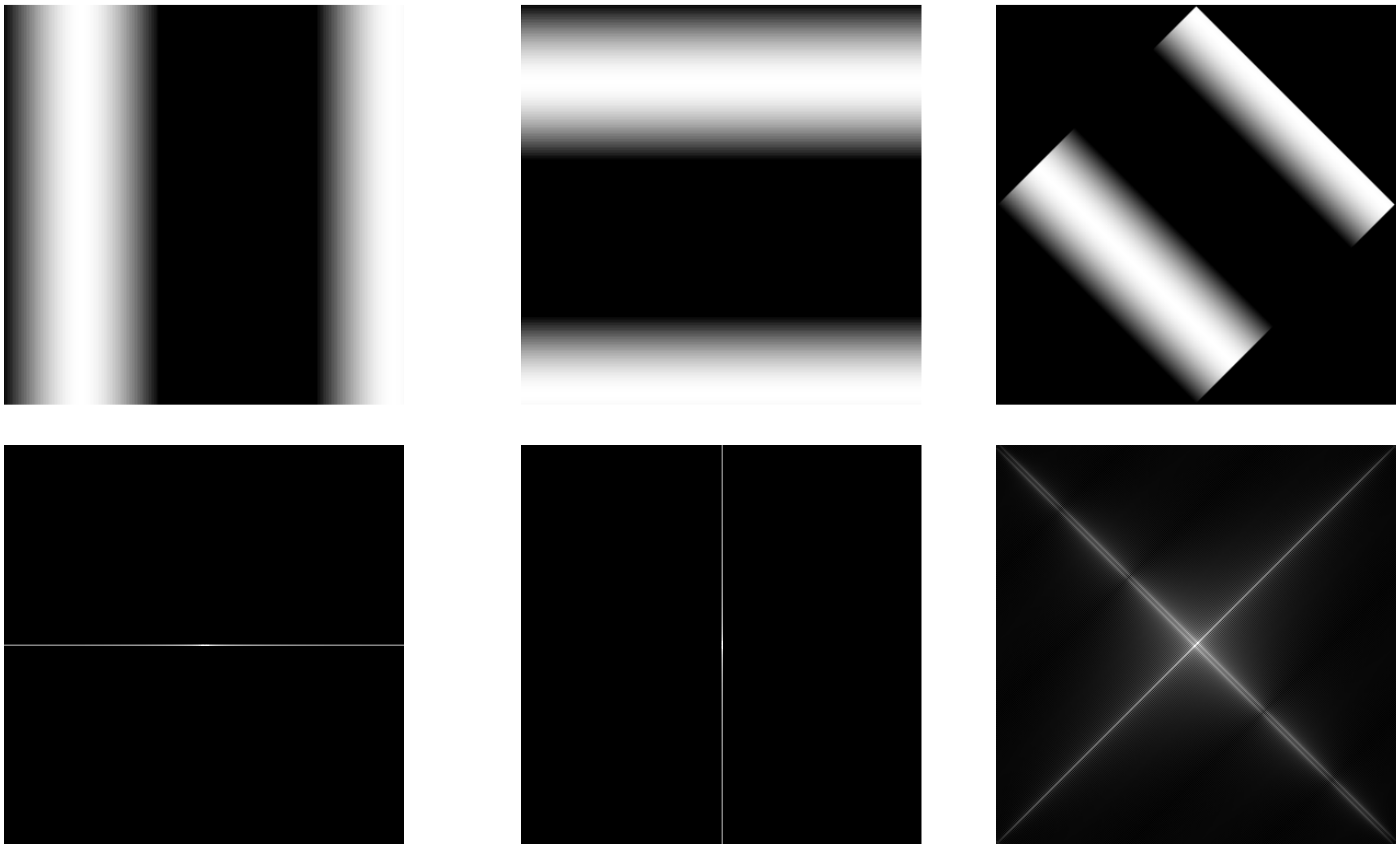
  - Por tanto, gracias a esta propiedad de simetría, para calcular la magnitud de los puntos de un periodo completo, tan sólo necesitamos calcular los N/2+1 primeros puntos, siempre y cuando el origen de la transformada este centrado en el punto (N/2,N/2). Para conseguir este movimiento del origen en la transformada, podemos aplicar la propiedad de traslación

1.1. Transformada de Fourier

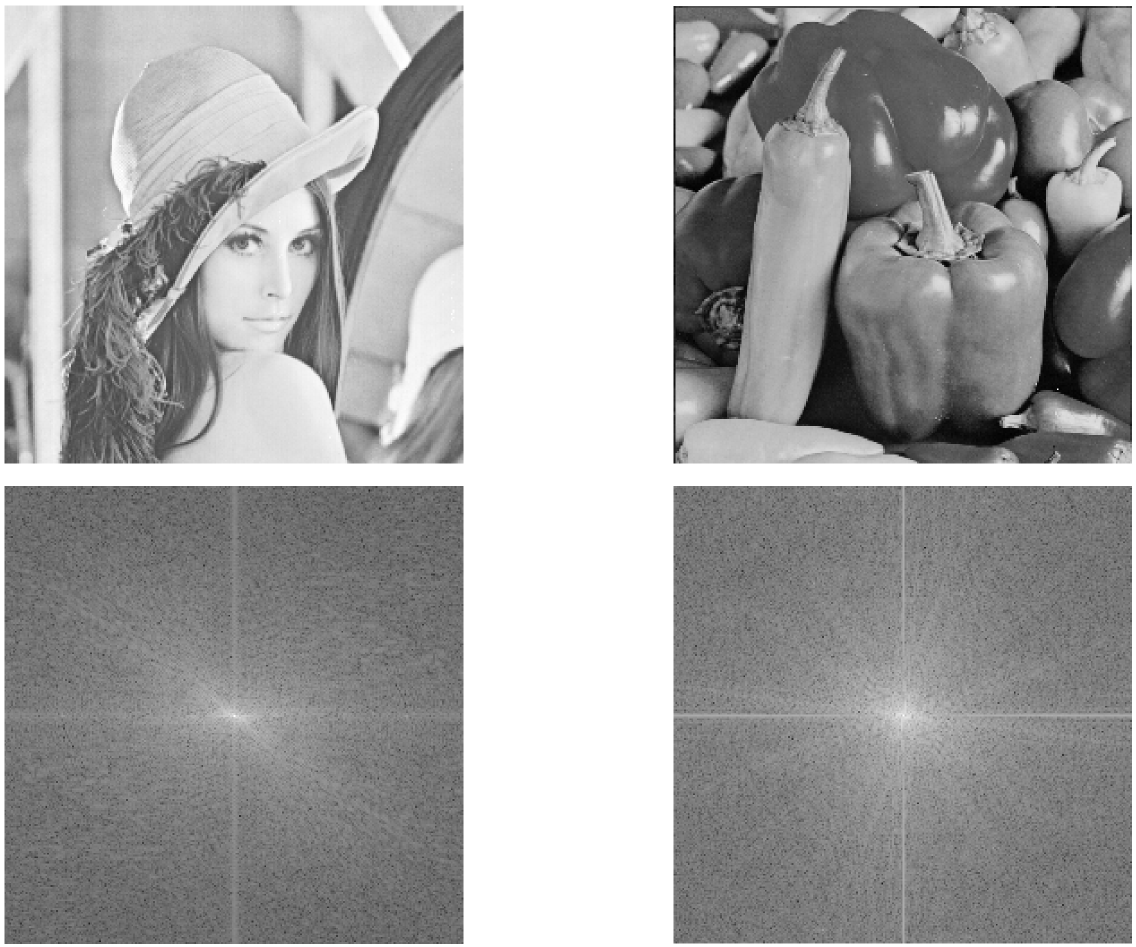
- Propiedades Transformada de Fourier Discreta bidimensional (TFD)

- La rotación:
  - Si rotamos la función f(x,y) un ángulo determinado, la transformada de Fourier también será afectada por una rotación del mismo ángulo. Esta propiedad también se da a la inversa, es decir, si la transformada se rota en un determinado ángulo, la transformada inversa también se verá rotada ese mismo ángulo

1.1. Transformada de Fourier



1.1. Transformada de Fourier

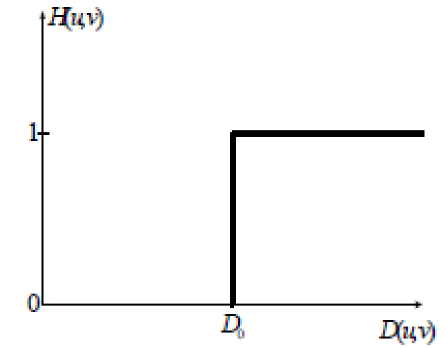


1.1. Transformada de Fourier

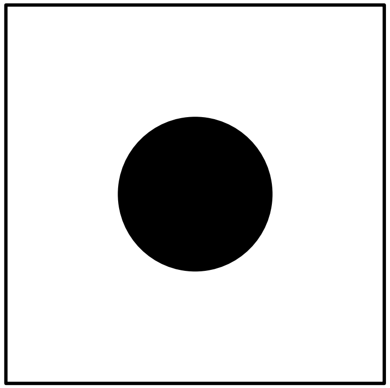
- Filtrado Paso Alto: dominio de la frecuencia

$$G(u,v) = H(u,v)\,F(u,v)$$

  - *H*(*u*,*v*): función de transferencia
  - *F*(*u*,*v*): transformada de Fourier de la imagen



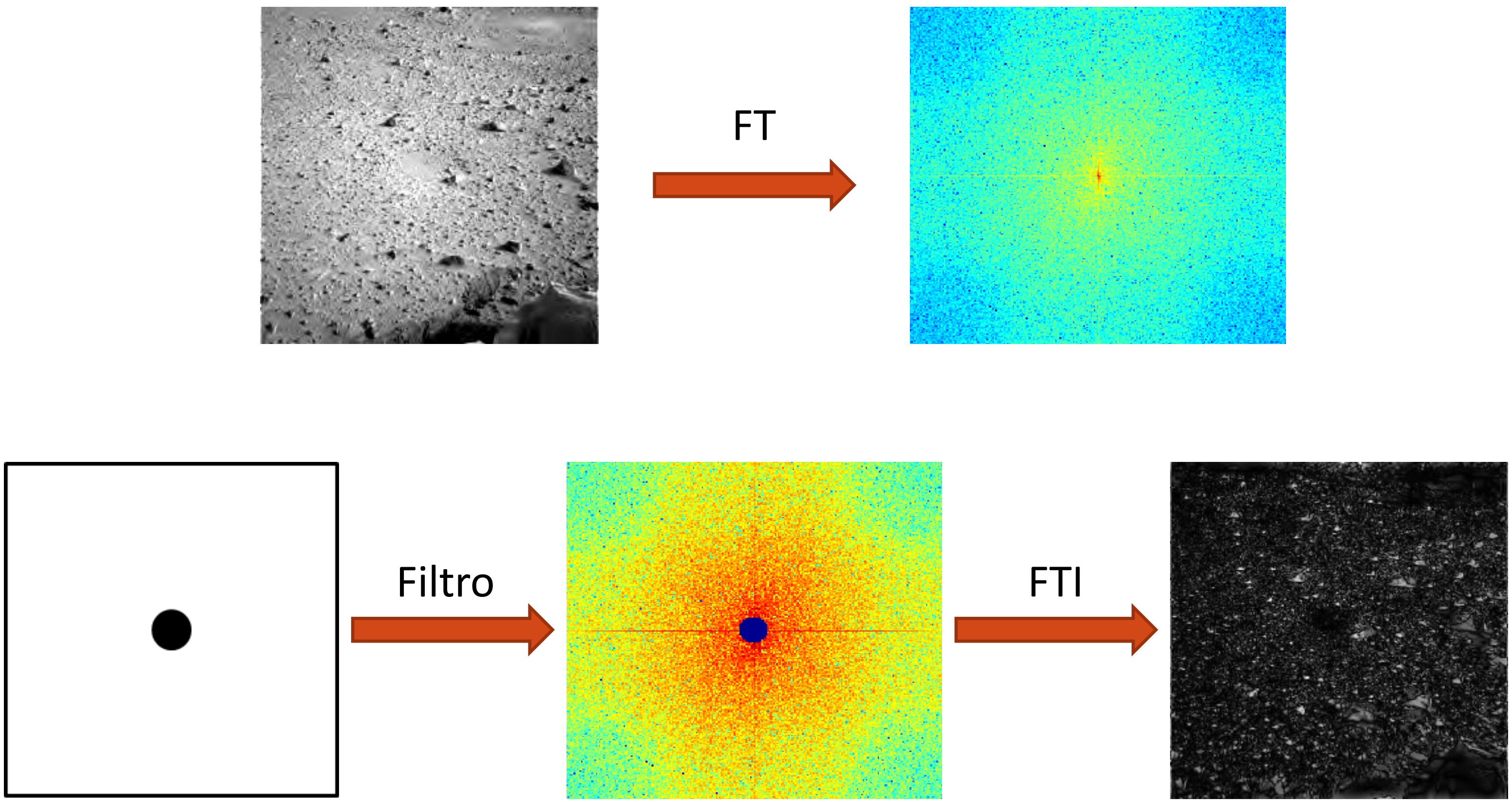
$$H(u,v) = \begin{cases} 0 \ si \ D(u,v) \leq D_0 \\ 1 \ si \ D(u,v) > D_0 \end{cases}$$



$$D(u,v) = \sqrt{u^2 + v^2}$$

- Este filtro conserva las altas frecuencias

1.1. Transformada de Fourier

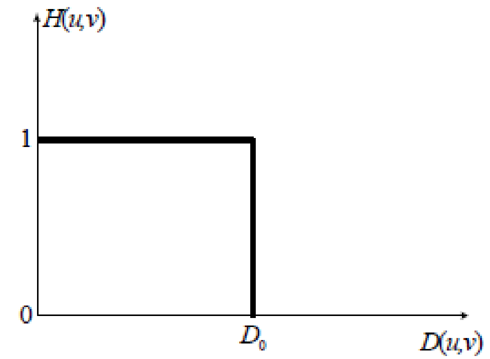


1.1. Transformada de Fourier

- Filtrado Paso Bajo: dominio de la frecuencia

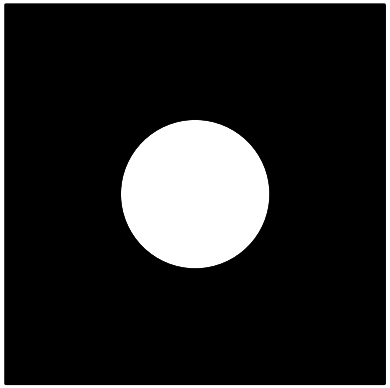
$$G(u,v) = H(u,v)\,F(u,v)$$

  - *H*(*u*,*v*): función de transferencia
  - *F*(*u*,*v*): transformada de Fourier de la imagen



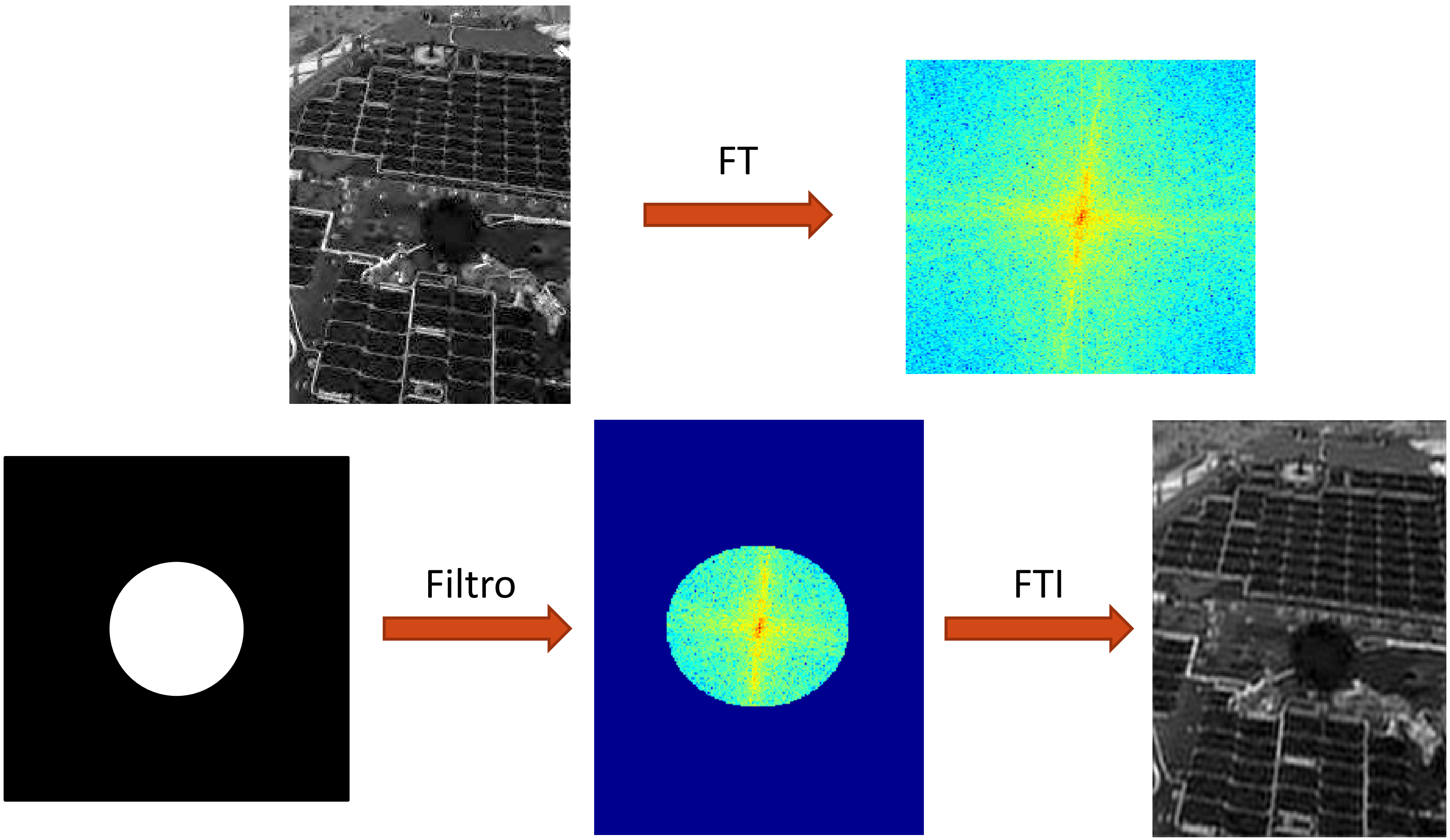
$$H(u,v) = \begin{cases} 1 \; si \; D(u,v) \leq D_0 \\ 0 \; si \; D(u,v) > D_0 \end{cases}$$

$$D(u,v) = \sqrt{u^2 + v^2}$$



- Este filtro conserva las bajas frecuencias

1.1. Transformada de Fourier

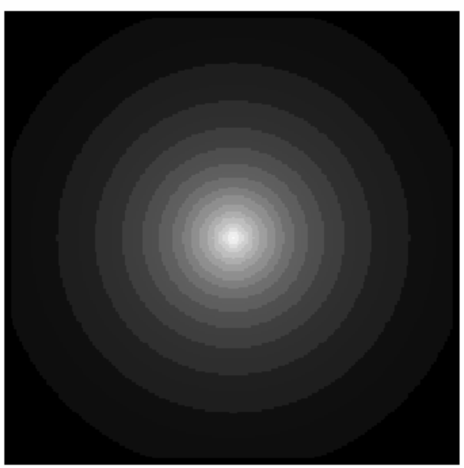


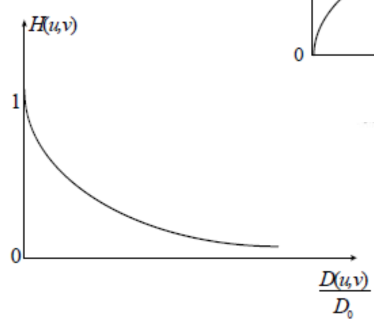
1.1. Transformada de Fourier

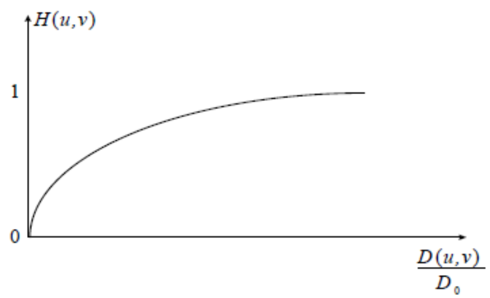
- Hay distintos tipos de filtros: $G(u,v) = H(u,v)\,F(u,v)$

- Butterworth: $H(u,v) = \dfrac{1}{1 + [D(u,v)/D_0]^{2n}}$

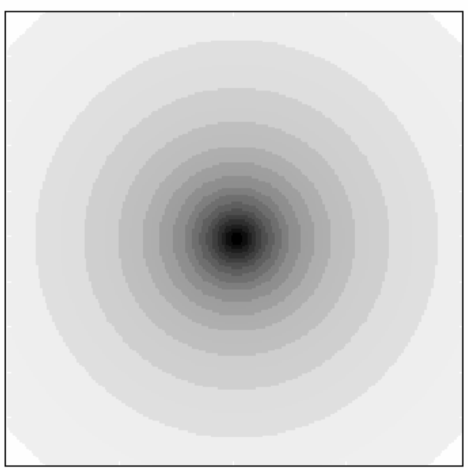
Paso Bajo





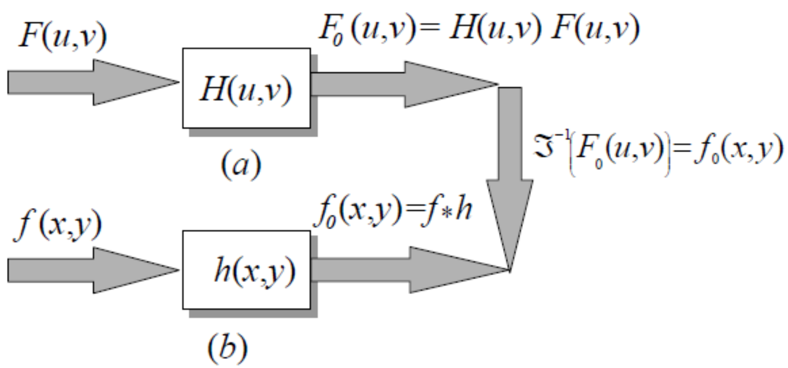


Paso Alto

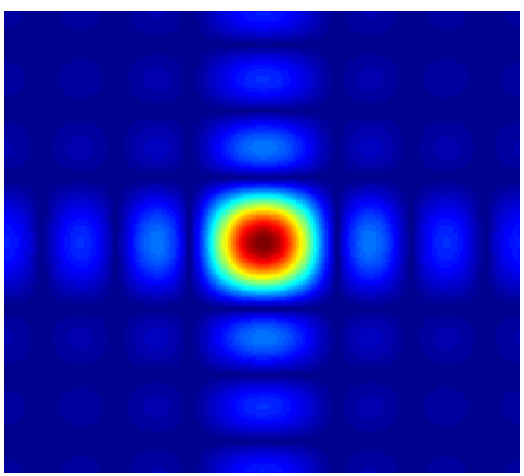


1.1. Transformada de Fourier

- Máscaras:
- Convolución:



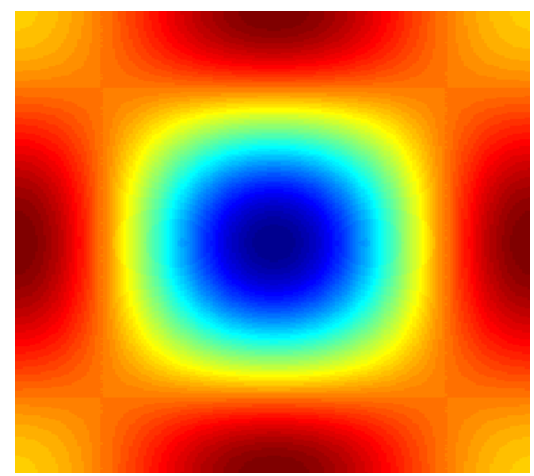
Paso Bajo



$$h \equiv \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Paso Alto

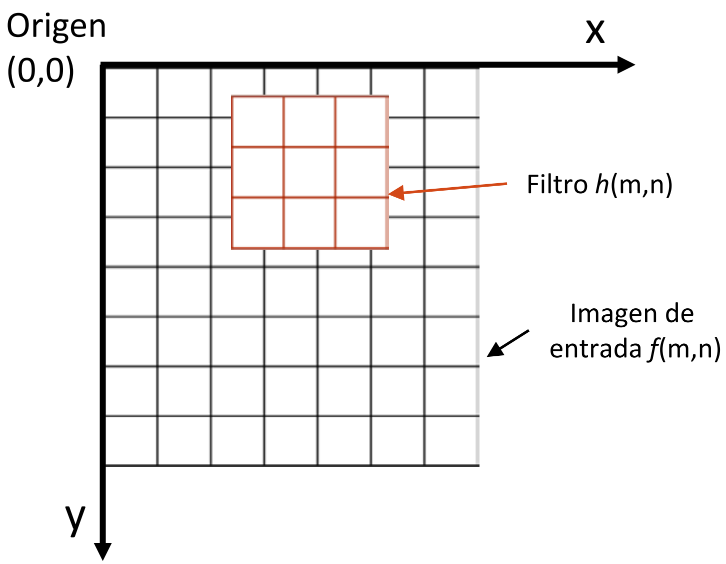
$$h \equiv \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



1.1. Transformada de Fourier

- Convolución:

$$f_0 = f * h = \sum_m \sum_n f(m,n)h(x-m,y-n) = \sum_m \sum_n f(x-m,y-n)h(m,n)$$

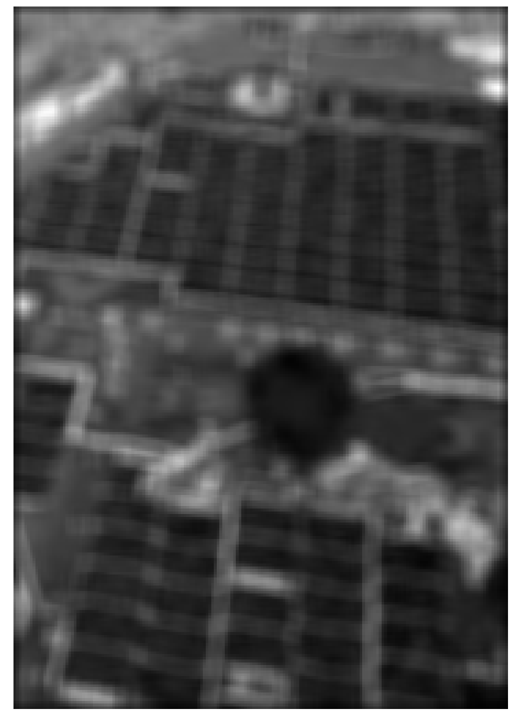


$h(-1,-1)$ 84 $f(x-1,y-1)$	$h(-1,0)$ 220 $f(x-1,y)$	$h(-1,+1)$ 100 $f(x-1,y+1)$
$h(0,-1)$ 55 $f(x,y-1)$	$h(0,0)$ 125 $f(x,y)$	$h(0,+1)$ 202 $f(x,y+1)$
$h(+1,-1)$ 185 $f(x+1,y-1)$	$h(+1,0)$ 68 $f(x+1,y)$	$h(+1,+1)$ 142 $f(x+1,y+1)$

1.1. Transformada de Fourier

- Convolución:

$$h \equiv \frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



1.1. Transformada de Fourier

- Convolución:

$$h = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Índice de contenidos

1.2. Transformada del coseno

- La transformada de coseno discreta (DCT del inglés Discrete Cosine Transform, 1974) es una transformada basada en la Transformada de Fourier discreta, pero utilizando únicamente números reales
- Características útiles para la compresión de imágenes:
  - La transformada de coseno tiene excelentes propiedades de compactación de energía, lo que significa que concentra casi toda la energía en los primeros coeficientes de la transformada
  - La transformación es independiente de los datos. El algoritmo aplicado no varia con los datos que recibe
  - Hay fórmulas para el cálculo rápido del algoritmo, como podría ser la FFT para la DFT
  - Produce pocos errores en los límites de los bloques imagen
  - Tiene una interpretación frecuencial de los componentes transformados que permite aprovechar al máximo la capacidad de compresión

1.2. Transformada del coseno

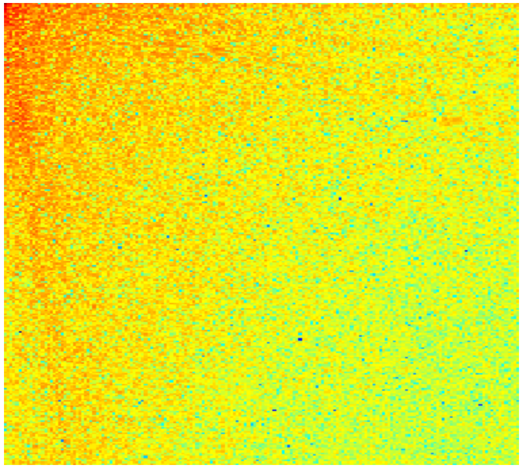
$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y) cos\left[\frac{(2x+1)\pi u}{2N}\right] cos\left[\frac{(2y+1)\pi v}{2N}\right]$$

$u, v = 0,1,2,\ldots,N\text{-}1$

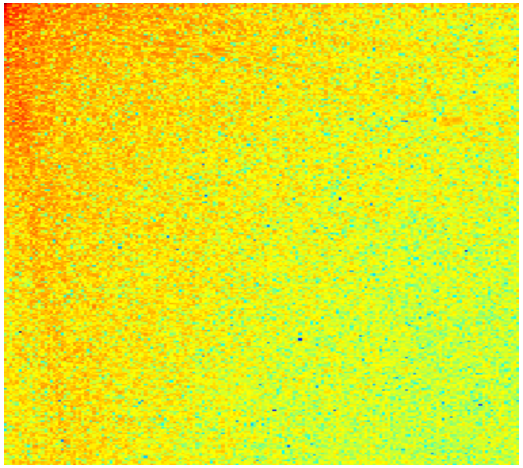
$$\alpha(u),\alpha(v) = \begin{cases} \sqrt{\dfrac{1}{N}} & \text{para } u,v=0 \\ \sqrt{\dfrac{2}{N}} & \text{para } u,v=1,2,\ldots,N-1 \end{cases}$$

1.2. Transformada del coseno

DCT

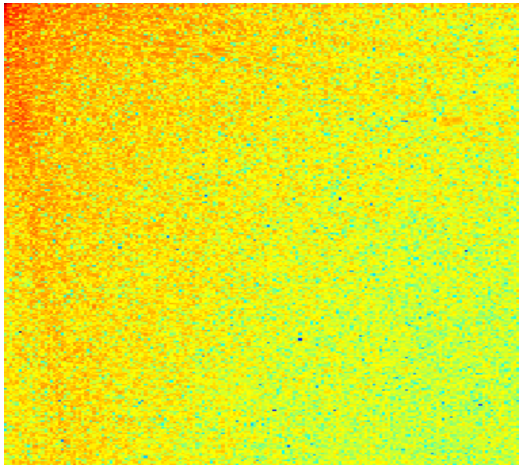


1.2. Transformada del coseno

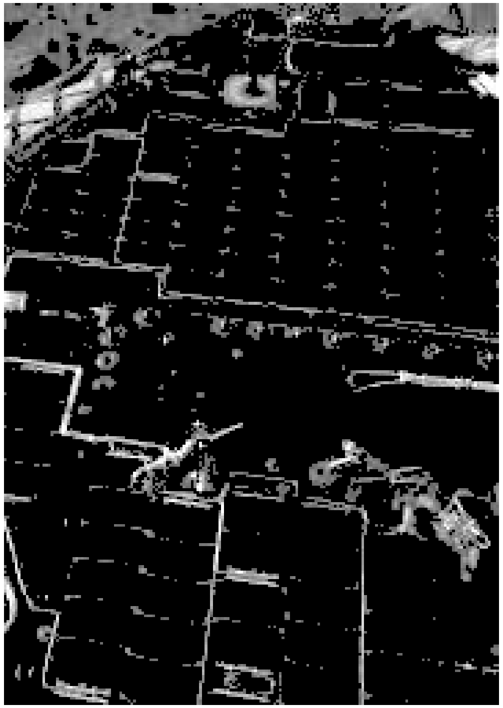


DCT
Inversa

1.2. Transformada del coseno



DCT
Inversa



Eliminando coeficientes cuyo valor absoluto < 100 (se asignan a 0)

1.2. Transformada del coseno

- Compresión. Lo que hace a JPEG completamente diferente de otros formatos (PNG, GIF, TIFF,...) es el hecho de aceptar la pérdida de información a la hora de comprimir
- Este tipo de compresión consta de 3 pasos:
  1. Primero se pasa la imagen del formato **RGB al formato YIQ**. El formato de color YIQ representa una división entre la luminosidad (cantidad de luz percibida) y la información sobre el color. El ojo humano es más sensible a la luminosidad que al color, cosa que se aprovecha para la compresión
  2. Después, se realiza una transformación en la imagen mediante la **transformada discreta del coseno** ya que permite comprimir la imagen según determinados patrones de frecuencia. Se trata de un método de compresión **con pérdida** de datos
  3. Por último, se **codifica el conjunto de datos** obtenidos al aplicar la TDC, usando un **método que no producen pérdida** (código de Huffman)
- Para recuperar la imagen se usa el proceso inverso de transformación de imágenes

Índice de contenidos

1.3. Transformada de Wavelets

- Transformada directa:

	Haar	Daubechies
Paso Bajo	$\frac{1}{\sqrt{2}}[1,1]$	$\frac{1}{4\sqrt{2}}[1-\sqrt{3}, 3-\sqrt{3}, 3+\sqrt{3}, 1+\sqrt{3}]$
Paso Alto	$\frac{1}{\sqrt{2}}[-1,1]$	$\frac{1}{4\sqrt{2}}[-1-\sqrt{3}, 3+\sqrt{3}, -3+\sqrt{3}, 1-\sqrt{3}]$

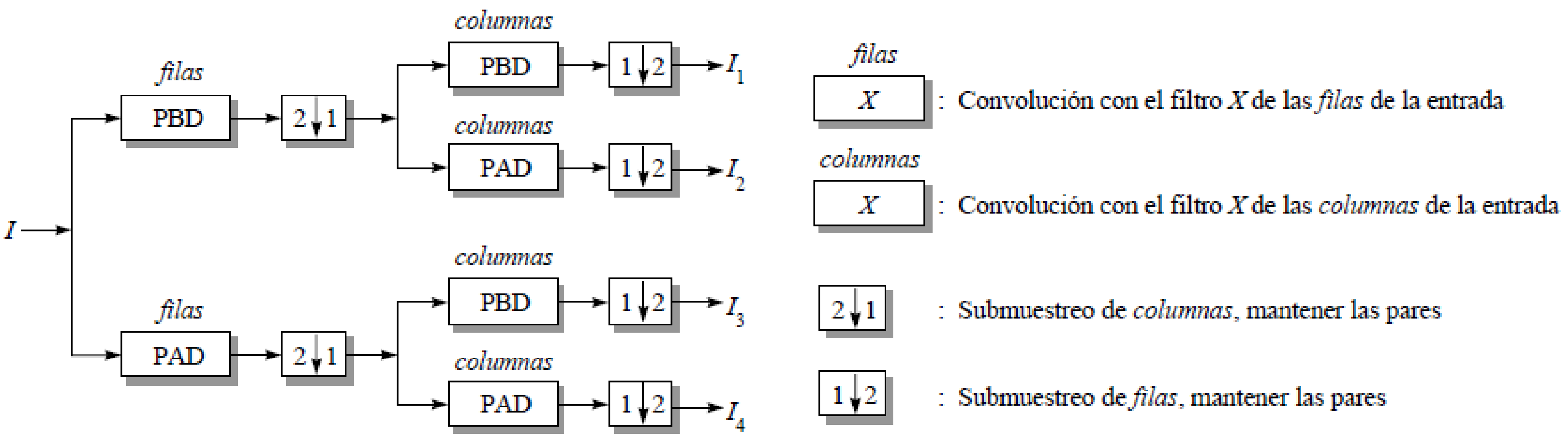
- Otras familias de Wavelets: Mahavir, Symlets, Meyer, etc.

1.3. Transformada de Wavelets

- **Paso 1:** Realizar la convolución de las filas con el filtro paso bajo y guardar los resultados
- **Paso 2:** Realizar la convolución de las columnas con el filtro paso bajo, a partir de los resultados del paso 1. Obtener una imagen reducida tomando sólo un píxel de cada dos; esto nos genera una versión que denominamos ***paso bajo/paso bajo*** de la imagen
- **Paso 3:** Realizar la convolución del resultado del paso 1 con el filtro paso alto en las columnas. Obtener una imagen reducida tomando sólo un píxel de cada dos, obteniendo ahora una imagen ***paso bajo/paso alto***
- **Paso 4:** Realizar la convolución de la imagen original con el filtro paso alto en las filas y guardar el resultado
- **Paso 5:** Realizar la convolución del resultado del paso 4 con el filtro paso bajo en las columnas. Obtener una imagen reducida tomando sólo un píxel de cada dos, obteniendo ahora una imagen ***paso alto/paso bajo***
- **Paso 6:** Realizar la convolución de las columnas del resultado del paso 4 con el filtro paso alto. Obtener una imagen reducida tomando sólo un píxel de cada dos, obteniendo ahora una imagen ***paso alto/paso alto***

1.3. Transformada de Wavelets

- Transformada directa:



1.3. Transformada de Wavelets

- Transformada directa:

Paso Bajo / Paso Bajo	Paso Bajo / Paso Alto
Paso Alto / Paso Bajo	Paso Alto / Paso Alto

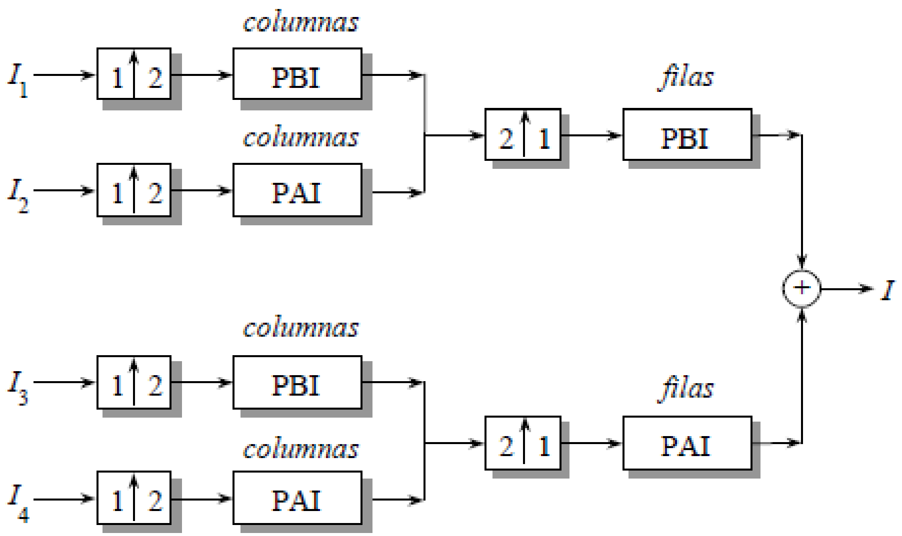
1.3. Transformada de Wavelets

- Transformada inversa:

	Haar	Daubechies
Paso Bajo	$\frac{1}{\sqrt{2}}[1,1]$	$\frac{1}{4\sqrt{2}}[1+\sqrt{3}, 3+\sqrt{3}, 3-\sqrt{3}, 1-\sqrt{3}]$
Paso Alto	$\frac{1}{\sqrt{2}}[1,-1]$	$\frac{1}{4\sqrt{2}}[1-\sqrt{3}, -3+\sqrt{3}, 3+\sqrt{3}, -1-\sqrt{3}]$

1.3. Transformada de Wavelets

- Transformada inversa:



filas $\boxed{X}$: Convolución con el filtro X de las *filas* de la entrada

columnas $\boxed{X}$: Convolución con el filtro X de las *columnas* de la entrada

$\boxed{2 \uparrow 1}$: Extender imagen, dada una imagen con m *columnas*, insertar m-1 columnas de ceros entre las m columnas dadas

$\boxed{1 \uparrow 2}$: Extender imagen, dada una imagen con n *filas*, insertar n-1 filas de ceros entre las n filas dadas

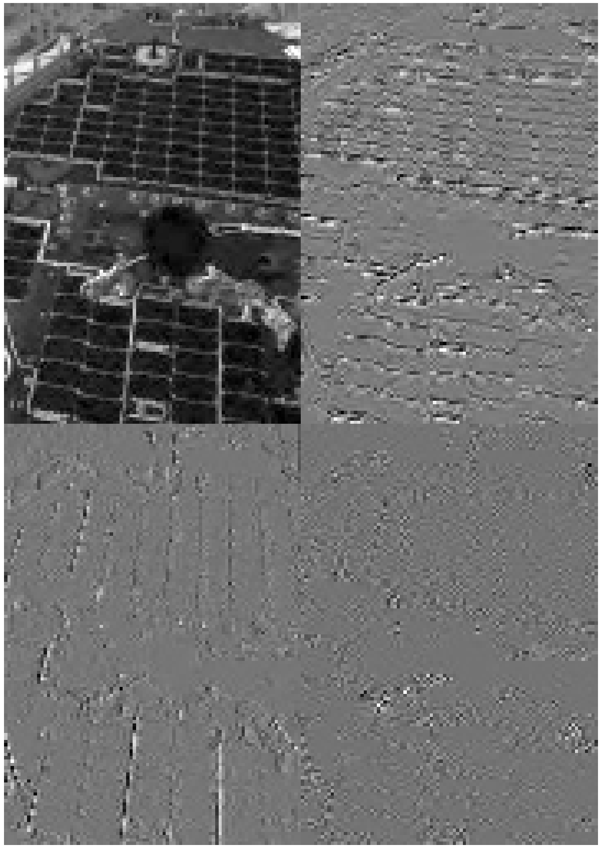
$\oplus$: Cada imagen de entrada I_i genera una imagen extendida por filas y columnas siguiendo el camino correspondiente. Si la imagen extendida es de mayor dimensión que la de la imagen I, se elige sólo la parte central de la imagen extendida cuya dimensión coincide exactamente con la de I. Las cuatro imágenes resultantes se suman para obtener I

1.3. Transformada de Wavelets

- Transformada directa: descomposición nivel 1

Coeficientes de aproximación

Coeficientes de detalle horizontales



Coeficientes de detalle verticales

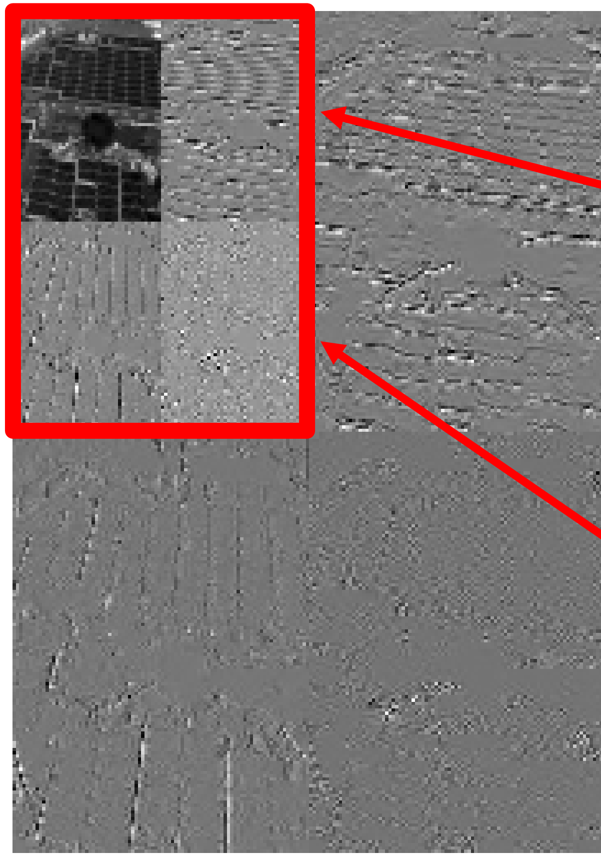
Coeficientes de detalle diagonales

1.3. Transformada de Wavelets

- Transformada directa: descomposición nivel 2

Coeficientes de aproximación

Coeficientes de detalle horizontales



Coeficientes de detalle verticales

Coeficientes de detalle diagonales

Universidad Rey Juan Carlos | Escuela Técnica Superior Ingeniería de Telecomunicación

Índice de contenidos

1.4. Ejemplos: fourier

```
int main(int argc, char ** argv)
{
    help(argv);
    const char* filename = argc >=2 ? argv[1] : "lena.jpg";
    Mat I = imread( samples::findFile( filename ), IMREAD_GRAYSCALE);
    if( I.empty()){
        cout << "Error opening image" << endl;
        return EXIT_FAILURE;
    }

    // 1. Expand the image to an optimal size.
    Mat padded;
    int m = getOptimalDFTSize( I.rows );
    int n = getOptimalDFTSize( I.cols ); // on the border add zero values
    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));

    // 2. Make place for both the complex and the real values
    Mat planes[] = {Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F)};
    Mat complexI;
    merge(planes, 2, complexI);         // Add to the expanded a

    // 3. Make the Discrete Fourier Transform
    dft(complexI, complexI);            // this way the result may fit
    //…
```

Este ajuste en el tamaño es para mejorar el rendimiento del algoritmo

El resultado de la transformada de Fourier es complejo

Se calcula la Transformada de Fourier Discreta (DFT en inglés) a través de dft

Universidad Rey Juan Carlos | Escuela Técnica Superior Ingeniería de Telecomunicación

Fork me on GitHub

1.4. Ejemplos: fourier

```
    // 4. Transform the real and complex values to magnitu
    // logarithmic scale => log(1 + sqrt(Re(DFT(I))^2 + Im
    split(complexI, planes);                     // plan
    magnitude(planes[0], planes[1], planes[0]);// pla
    Mat magI = planes[0];

    // 5. Switch to a logarithmic scale
    magI += Scalar::all(1);                      // switch t
    log(magI, magI);

    // 6. Crop and rearrange  // crop the spectrum, if it
    magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

    // rearrange the quadrants of Fourier image  so that the origin is at the image center
    int cx = magI.cols/2;
    int cy = magI.rows/2;
    Mat q0(magI, Rect(0, 0, cx, cy));   // Top-Left - Create a ROI per quadrant
    Mat q1(magI, Rect(cx, 0, cx, cy));  // Top-Right
    Mat q2(magI, Rect(0, cy, cx, cy));  // Bottom-Left
    Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-Right
    Mat tmp;                           // swap quadrants (Top-Left with Bottom-Right)
    q0.copyTo(tmp);    q3.copyTo(q0);    tmp.copyTo(q3);
    q1.copyTo(tmp);                    // swap quadrant (Top-Right with Bottom-Left)
    q2.copyTo(q1);
    tmp.copyTo(q2);
//…
```

Un número complejo tiene una parte real (Re) y una compleja (imaginaria - Im). Los resultados de una DFT son números complejos. La magnitud de una DFT es:

$$M = \sqrt[2]{Re(DFT(I))^2 + Im(DFT(I))^2}$$

Fork me on GitHub

1.4. Ejemplos: fourier

```
    // 4. Transform the real and complex values to magnitu
    // logarithmic scale => log(1 + sqrt(Re(DFT(I))^2 + Im
    split(complexI, planes);                   // planes[0
    magnitude(planes[0], planes[1], planes[0]);// planes[0
    Mat magI = planes[0];

    // 5. Switch to a logarithmic scale
    magI += Scalar::all(1);
    log(magI, magI);

    // 6. Crop and rearrange  // crop the spectrum, if it
    magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

    // rearrange the quadrants of Fourier image  so that t
    int cx = magI.cols/2;
    int cy = magI.rows/2;
    Mat q0(magI, Rect(0, 0, cx, cy));   // Top-Left - Crea
    Mat q1(magI, Rect(cx, 0, cx, cy));  // Top-Right
    Mat q2(magI, Rect(0, cy, cx, cy));  // Bottom-Left
    Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-Righ
    Mat tmp;                           // swap quadrants (
    q0.copyTo(tmp);    q3.copyTo(q0);    tmp.copyTo(q3);
    q1.copyTo(tmp);                    // swap quadrant (Top-Right with Bottom-Left)
    q2.copyTo(q1);
    tmp.copyTo(q2);
//…
```

Como el rango de los coeficientes de Fourier es demasiado alto para ser mostrado en pantalla, y debido a que algunos valores son pequeños y otros altos, cambiamos la escala lineal a una logarítmica

Se eliminan los valores introducidos en el primer paso, y se reorganizan los cuadrantes para visualizarlo, de modo que el origen (0,0) corresponde con el centro de la imagen

1.4. Ejemplos: fourier

```
    // 7. Normalize
    normalize(magI, magI, 0, 1, NORM_MINMAX); // Tra                          into a
                                          // viewabl                         0 and 1).
    // 8. Results
    imshow("Input Image"       , I   );     // Show th
    imshow("spectrum magnitude", magI);

    // 9. Calculating the idft
    Mat inverseTransform;
    idft(complexI, inverseTransform, cv::DFT_INVERSE|cv::DFT_REAL_OUTPUT);
    normalize(inverseTransform, inverseTransform, 0, 1, NORM_MINMAX);
    imshow("Reconstructed", inverseTransform);

    // 10. Creating idft from spectrum
    //doSomethingWithTheSpectrum();

    // IFFT
    Mat inverseTransform2;
    dft(magI, inverseTransform2, cv::DFT_INVERSE|cv::DFT_REAL_OUTPUT);

    // Back to 8-bits
    Mat finalImage;
    inverseTransform2.convertTo(finalImage, CV_8U);
    imshow("Inverse transform from spectrum", inverseTransform);
    waitKey();
    return EXIT_SUCCESS;
}
```

Se normalizan los valores para visualización

Para calcular la inversa de la transformada, utilizamos la función idft aplicada a los valores complejos obtenidos en el paso 3

En el caso de calcular la inversa a partir del espectro, necesitamos utilizar la función dft indicando que realice la inversa

Posteriormente se convierte a una imagen en escala de grises

1.4. Ejemplos: coseno

```
int main() {
    // Read image
    Mat src = imread("../../images/lenna.jpg", 0);
    if(src.empty()) {
        cout << "the image is not exist" << endl;
        return -1;
    }
    resize(src, src, Size(512, 512));
    src.convertTo(src, CV_32F, 1.0/255);

    // Discrete Cosine Transform
    Mat srcDCT;
    dct(src, srcDCT);

    // Inverse Discrete Cosine Transform
    Mat InvDCT;
    idct(srcDCT, InvDCT);

    // Show images
    imshow("src", src);
    imshow("dct", srcDCT);
    imshow("idct", InvDCT);
    waitKey();

    return 0;
}
```

En este ejemplo se redimensiona la imagen

Para crear la Transformada Discreta del Coseno (DCT en inglés) se utiliza la función dct proporcionada por OpenCV

Para obtener la Transformada Discreta Inversa del Coseno (IDCT en inglés) se utiliza la función idct proporcionada por OpenCV

Índice de contenidos

2. Transformación espacial

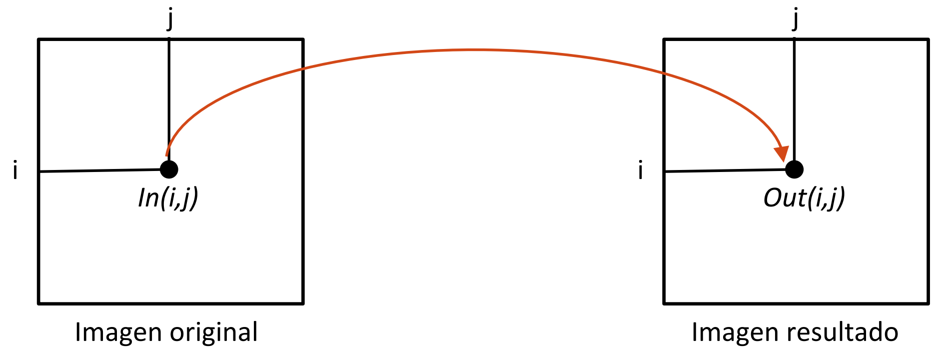
- Consiste en modificar el contenido de la misma con un objetivo concreto, como puede ser el de prepararla para un posterior análisis
- Para las **transformaciones píxel a píxel** generalmente se aplica una función sobre el valor de intensidad de cada píxel individualmente
- Las **transformaciones de vecindad** son aquellas en las que los píxeles vecinos intervienen en la misma con distintas finalidades, tales como eliminar el ruido presente en la imagen para así suavizarla o con el fin de extraer bordes
- Las **transformaciones lógicas** se aplican considerando que los valores numéricos de las imágenes se pueden representar a nivel de bits
- Las **transformaciones geométricas** modifican las coordenadas espaciales de la imagen, ofreciendo aspectos de la misma bajo diferentes resoluciones, siendo en ocasiones necesario modificar también los valores de intensidad de los píxeles de la imagen original

Índice de contenidos

1. Transformación del dominio (o frecuencia):
   - Transformada de Fourier
   - Transformada del coseno
   - Transformada de Wavelets
   - Ejemplos
2. Transformación espacial:
   - Transformación píxel a píxel
   - Transformaciones de vecindad
   - Transformaciones lógicas
   - Transformaciones geométricas
   - Ejemplos

2.1. Transformación píxel a píxel

- Transforma la imagen mediante la modificación uno a uno de los píxeles de la imagen
- La imagen obtenida tendrá el mismo tamaño que la original



- La transformación se obtiene a partir de la siguiente ecuación:

$$Out(i,j) = f(In(i,j))$$

- Donde *f* puede ser lineal o no

2.1. Transformación píxel a píxel

- Algunos de los principales operadores son:

Operador identidad:	$Out(i,j) = In(i,j)$
Operador inverso:	$Out(i,j) = 255 - In(i,j)$
Operador umbral:	$Out(i,j) = \begin{cases} 0 & si \;\; In(i,j) \leq p \\ 255 & si \;\; In(i,j) > p \end{cases}$
Operador intervalo de umbral binario:	$Out(i,j) = \begin{cases} 0 & si \;\; p_1 < In(i,j) < p_2 \\ 255 & si \;\; In(i,j) \leq p_1 \; o \;\; In(i,j) \geq p_2 \end{cases}$
Operador umbral escala de grises:	$Out(i,j) = \begin{cases} In(i,j) & si \;\; p_1 < In(i,j) < p_2 \\ 255 & si \;\; In(i,j) \leq p_1 \; o \;\; In(i,j) \geq p_2 \end{cases}$

2.1. Transformación píxel a píxel

Imagen original

Imagen inverso

Imagen umbral p=90

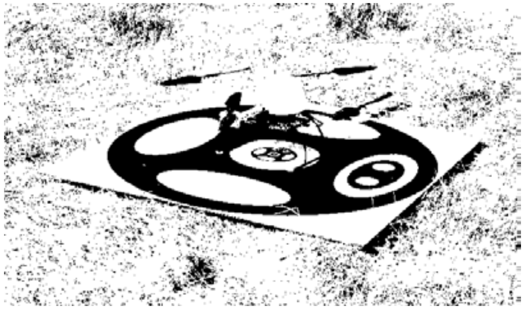


Imagen intervalo umbral binario p_1=50 y p_2=150

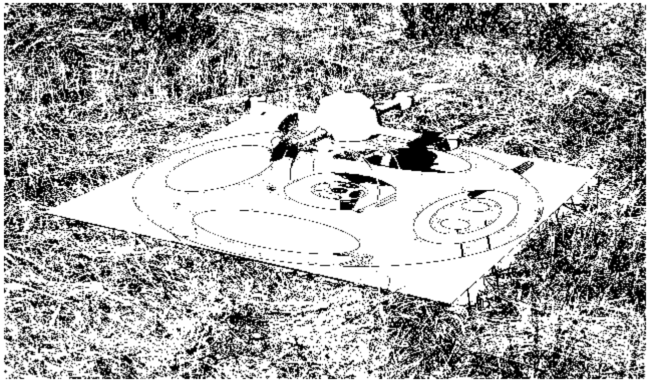
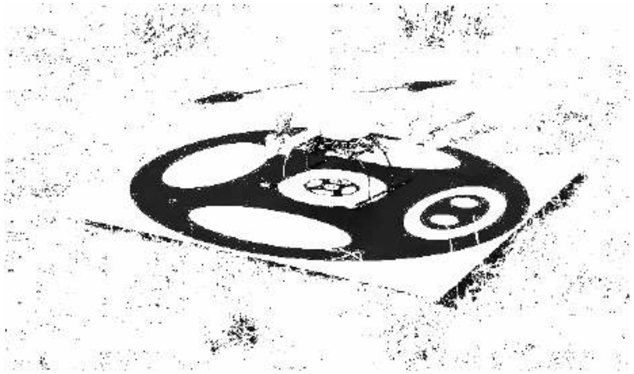


Imagen intervalo umbral escala grises p_1=1 y p_2=50



Índice de contenidos

1. Transformación del dominio (o frecuencia):
   - Transformada de Fourier
   - Transformada del coseno
   - Transformada de Wavelets
   - Ejemplos
2. Transformación espacial:
   - Transformación píxel a píxel
   - Transformaciones de vecindad
   - Transformaciones lógicas
   - Transformaciones geométricas
   - Ejemplos

2.2. Transformaciones de vecindad

- En las transformaciones de vecindad el valor del **píxel de salida** se obtiene tras realizar sobre el píxel de origen una **combinación de los valores de los píxeles vecinos**
- La transformación de la imagen se produce por la **combinación de píxeles**, en lugar de realizar una transformación píxel a píxel
- Con carácter general, el vecindario de un píxel lo componen ocho valores, correspondientes a las posiciones alrededor del píxel
- De manera que **el valor de intensidad del píxel de salida** Out(x,y) **es la suma promediada de los valores de intensidad de los ocho vecinos** alrededor del píxel de entrada In(x, y)
- La transformación de una imagen puede variar dependiendo de la influencia que cada uno de los vecinos ejerza. Esto se consigue mediante una **máscara** que permite escoger de manera selectiva los vecinos que intervienen en la transformación, y en qué medida contribuyen a la modificación del píxel central

2.2. Transformaciones de vecindad

- Convolución:

$$M1_{3x3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}; \quad M2_{3x3} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}; \quad M3_{3x3} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix};$$

$MX(-1,-1)$ 84 $In(x-1,y-1)$	$MX(-1,0)$ 220 $In(x-1,y)$	$MX(-1,+1)$ 100 $In(x-1,y+1)$
$MX(0,-1)$ 55 $In(x,y-1)$	$MX(0,0)$ 125 $Out(x,y)$	$MX(0,+1)$ 202 $In(x,y+1)$
$MX(+1,-1)$ 185 $In(x+1,y-1)$	$MX(+1,0)$ 68 $In(x+1,y)$	$MX(+1,+1)$ 142 $In(x+1,y+1)$

$$\begin{gathered} Out(x,y)_{M1} = \\ 1 \cdot In(x-1,y-1) + 1 \cdot In(x-1,y) + 1 \cdot In(x-1,y+1) \\ + 1 \cdot In(x,y-1) + 1 \cdot In(x,y) + 1 \cdot In(x,y+1) \\ + 1 \cdot In(x+1,y-1) + 1 \cdot In(x+1,y) + 1 \cdot In(x+1,y+1) \end{gathered}$$

$$\begin{gathered} Out(x,y)_{M2} = \\ 1 \cdot In(x-1,y-1) + 2 \cdot In(x-1,y) + 1 \cdot In(x-1,y+1) \\ + 0 \cdot In(x,y-1) + 0 \cdot In(x,y) + 0 \cdot In(x,y+1) \\ -1 \cdot In(x+1,y-1) - 2 \cdot In(x+1,y) - 1 \cdot In(x+1,y+1) \end{gathered}$$

2.2. Transformaciones de vecindad

Original

M1

M2

M3

Universidad Rey Juan Carlos | Escuela Técnica Superior Ingeniería de Telecomunicación

Índice de contenidos

1. Transformación del dominio (o frecuencia):
   - Transformada de Fourier
   - Transformada del coseno
   - Transformada de Wavelets
   - Ejemplos
2. Transformación espacial:
   - Transformación píxel a píxel
   - Transformaciones de vecindad
   - Transformaciones lógicas
   - Transformaciones geométricas
   - Ejemplos

2.3. Transformaciones lógicas

- Una imagen digital en gris no es ni más ni menos que una matriz de números
- Se pueden aplicar las mismas operaciones aritméticas que sobre matrices numéricas, con las limitaciones derivadas del hecho de que los valores de intensidad sólo toman valores en el rango establecido, generalmente en el intervalo [0, 255], en este caso por su representación de ocho bits
- Dado que la representación interna de los datos en un ordenador se realiza finalmente mediante una representación binaria, es posible aplicar operaciones lógicas binarias sobre estos datos
- Las operaciones lógicas habituales son: *and*, *or*, *xor*, *not* y derivadas

2.3. Transformaciones lógicas

- Partiendo de dos imágenes estereoscópicas, donde la misma escena ha sido capturada con dos cámaras ligeramente desplazadas horizontalmente

- Se pueden aplicar las transformaciones píxel a píxel que hemos visto para obtener imágenes binarias (con valores de 0 y 1)

2.3. Transformaciones lógicas

- Imágenes binarias con p=100

- Resultado de aplicar la negación lógica

2.3. Transformaciones lógicas

- OR, AND y XOR aplicadas sobre las negaciones lógicas:

- Se pueden aplicar otro tipo de operaciones relacionales tales como:

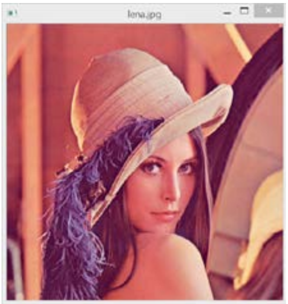
  $<$ $>$ $\leq$ $\geq$

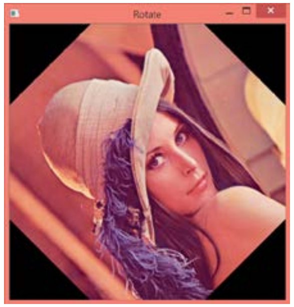
Índice de contenidos

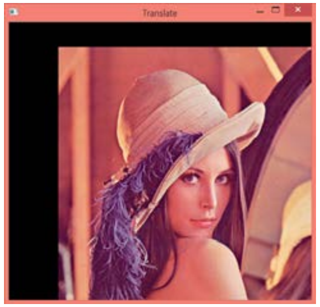
1. Transformación del dominio (o frecuencia):
   - Transformada de Fourier
   - Transformada del coseno
   - Transformada de Wavelets
   - Ejemplos
2. Transformación espacial:
   - Transformación píxel a píxel
   - Transformaciones de vecindad
   - Transformaciones lógicas
   - Transformaciones geométricas
   - Ejemplos

2.4. Transformaciones geométricas

- Los píxeles de la imagen resultado sufren un cambio en cuanto a la posición que ocupan en la imagen original
- Dado que los píxeles ocupan una posición discreta, no hay valores intermedios. Esto provoca que sea necesario aplicar **técnicas de interpolación**, de manera que una transformación genere nuevos valores en función de los ya existentes
- Las transformaciones elementales entre otras son:
  - Escalado
  - Rotación
  - Traslación







- Todo esto lo veremos en detalle en el próximo tema

Índice de contenidos

1. Transformación del dominio (o frecuencia):
   - Transformada de Fourier
   - Transformada del coseno
   - Transformada de Wavelets
   - Ejemplos
2. Transformación espacial:
   - Transformación píxel a píxel
   - Transformaciones de vecindad
   - Transformaciones lógicas
   - Transformaciones geométricas
   - Ejemplos

2.5. Ejemplos: pixel a pixel

```
int main() { //...
  // 1. Method inverse
  Mat dst1(src.rows, src.cols, src.type());
  // Read pixel values
  for ( int i=0; i<src.rows; i++ ) {
    for ( int j=0; j<src.cols; j++ ) {
      // You can now access the pixel value and calculate the new value
      dst1.at<uchar>(i,j) = (uint)(255 - (uint)src.at<uchar>(i,j));
    }
  }

  // 2. Method threshold
  Mat dst2(src.rows, src.cols, src.type());
  uint threshold_p = 150;
  // Read pixel values
  for ( int i=0; i<src.rows; i++ ) {
    for ( int j=0; j<src.cols; j++ ) {
      // You can now access the pixel value and calculate the new value
      uint value = (uint)(255 - (uint)src.at<uchar>(i,j));
      if (value > threshold_p)
        dst2.at<uchar>(i,j) = (uint)255;
      else
        dst2.at<uchar>(i,j) = (uint)0;
    }
  }
   //...
}
```

Se leen los píxeles uno a uno y se tratan de forma independiente:

1. Se aplica el inverso al valor del píxel
2. Se calcula el valor del píxel de forma binaria a partir de un umbral p = 150

2.5. Ejemplos: vecindad

```
int main() {
    // Read image
    Mat src = imread("../../images/lenna.jpg", 0);
    if(src.empty()) {
        cout << "the image is not exist" << endl;
        return -1;
    }
    resize(src, src, Size(512, 512));
    src.convertTo(src, CV_32F, 1.0/255);

    // Masks
    Mat M1 = (Mat_<char>(3,3) <<  1, 1, 1,
                                  1, 1, 1,
                                  1, 1, 1);

    // Applying masks
    Mat dst1, dst2, dst3;
    filter2D( src, dst1, src.depth(), M1 );

    // Show images
    imshow("Original", src);
    imshow("Mask 1", dst1);
    waitKey();

    return 0;
}
```

Se crea la máscara con el tamaño deseado y se inicializan los valores los cuales se multiplicarán por lo valores de cada píxel y se sumarán

Se utiliza la función filter2D la cual se encarga de aplicar la máscara a cada píxel teniendo en cuenta el tamaño de vecindad elegido por la máscara

Universidad Rey Juan Carlos | Escuela Técnica Superior Ingeniería de Telecomunicación

Fork me on GitHub

2.5. Ejemplos: lógica

```
int main() {
    // create two input matrices zeros
    Mat im1 = Mat::zeros( Size(400, 400), CV_8UC1);
    Mat im2 = Mat::zeros( Size(400, 400), CV_8UC1);
    // Draw a circle on images moving 10px in image 2
    circle(im1, Point(200, 200), 100.0, Scalar (255, 255, 255), 1, 8);
    circle(im2, Point(210, 200), 100.0, Scalar (255, 255, 255), 1, 8);

    // Display circles
    imshow("Circle 1", im1);
    imshow("circle 2", im2);

    // Output images
    Mat output1, output2;
    // Compute the bitwise AND of input images and sto
    bitwise_and(im1, im2, output1);
    // Compute the bitwise OR of input images and st
    bitwise_or(im1, im2, output2);

    // Display the output images
    imshow("bitwise_and", output1);
    imshow("bitwise_or", output2);

    waitKey(0);
    return 0;
}
```

Para este ejemplo se crean dos imágenes de 400x400 y se dibuja un círculo blanco en su interior

Se utiliza la función bitwise_and o bitwise_or las cuales reciben las dos imágenes a las que hay que aplicar la operación lógica y la imagen donde se almacenará el resultado