Monografía

Patrones de Moiré

Minerva Alvarez García
Neil Charles Bruce Davidson

23 de mayo de 2025

Proyecto de Docencia y Divulgación

Financiamiento interno del ICAT

Título: Patrones de Moiré

Autores: Minerva Alvarez García, Neil Charles Bruce Davidson

Afiliación: Grupo de Instrumentación Óptica, ICAT

Resumen

Este es una monografía que compila todo el trabajo hecho durante el servicio social: *Apoyo en los proyectos que se desarrollan en el ICAT*, “Patrones de Moiré para divulgación” con clave 2023-12/83-160, realizada por Minerva Alvarez García bajo la supervisión del Dr. Neil C. Bruce Davidson. En este trabajo se estudian los patrones Moiré formados por dos rejillas a blanco y negro desde el análisis de Fourier con el apoyo del software Matlab. Entre las rejillas usadas se encuentran: cosinusoidales, cuadradas, puntos y anillos. Adicionalmente, se escribieron programas para la producción de patrones arbitrarios que formen imágenes a blanco y negro, y en ciertos casos, simulen movimiento. Se incluyen los programas para el uso de lectores interesados en el tema. Los resultados y programas presentados sirven para preparar materiales de divulgación y para demostraciones en clases de la transformada de Fourier. Cabe destacar que los materiales producidos fueron utilizados en el evento del Día del Niño en el ICAT, con mucho éxito.

Índice

1. Introducción

Definimos los patrones de Moiré como los patrones formados al superponer dos o más rejillas, dichos patrones no necesariamente aparecen explícitamente en ninguna de las estructuras originales (Amidror, 2009). Se pueden estudiar desde el dominio de frecuencias usando el análisis de Fourier.

Dentro de la ciencia y la tecnología, los patrones de Moiré tienen diversas áreas de aplicaciones entre las cuales se encuentran: la metrología (Reid, 1984), microscopía (Gustafsson, 2000, Kobayashi, 2000), interferometría (Miao et al., 2016), magnificación de objetos periódicos utilizando arreglos de lentes (Stevens, 1999, Kamal et al., 1998), navegación (Bergkvist and Forsen, 1986, Kazda and Caves, 2015), autentificación de documentos y antifalsificación (Amidror, 2009), estudios del grafeno (Schouteden et al., 2015, MacDonald, 2019), entre otras cosas. Por otro lado, resultan atractivos en públicos jóvenes o en general en personas no tan relacionadas con la ciencia por lo que también son usados en ámbitos como el arte y la literatura (Seder, 2024).

Para trabajar con rejillas desde una aproximación espectral es necesario traducir imágenes 2D en un objeto al que pueda aplicarse la teoría de Fourier (Goodman, 1996). Por facilidad en este trabajo sólo se consideran imágenes a blanco y negro, sin embargo, existe teoría para patrones Moiré a color (Amidror, 2009).

Otras hipótesis que se ocupan en este trabajo para abordar los patrones de Moiré desde el análisis de Fourier son los siguientes. Los programas presentados aquí modelan la transmisión de la luz a través de las rejillas, pero se aproxima esta transmisión con el modelo geométrico de la luz. Esto significa que se desprecian los efectos de esparcimiento, o difracción. Además, se asume que las rejillas que forman el patrón se encuentran en contacto total; se omiten los aspectos cinéticos de los patrones, esto es, cualquier efecto que se manifieste debido a la velocidad con la que se muevan las rejillas superpuestas; por último, se toma en cuenta un modelo simple de la visión humana.

Ahora sí, dada una rejilla monocromática se le puede asociar una imagen en el dominio de imágenes de acuerdo con una función de reflectancia $r(\ \ )$ la cual asigne a cada punto (x, y) de la rejilla, un número entre 0 y 1 que represente la luz reflejada por la rejilla; bajo esta convención, 0 significa que no hay reflexión de luz, o sea, en este punto la rejilla es negra y, por el contrario, 1 significa que toda la luz es reflejada, es decir, la rejilla físicamente es blanca; cualquier número intermedio corresponde a alguna tonalidad de gris. Es conveniente esta forma de traducir rejillas en imágenes, pues intuitivamente al superponer físicamente rejillas se espera que, al juntar negro con negro, se obtenga negro; con la misma lógica, se espera que blanco con blanco dé blanco, por lo que naturalmente se llega a que la superposición de imágenes esté dada por una operación multiplicativa. De esta forma, se entiende que la superposición de m rejillas monocromáticas es la imagen:

$$r(x, y) = r_1(x, y) \cdot r_2(x, y) \cdot \ldots \cdot r_m(x, y), \tag{1}$$

donde $r_i(x, y)$ es el valor que asignó la función de reflectancia al punto (x, y) de la i-ésima rejilla, con $i = 1,2, \cdots, m$.

Denotando la transformada de Fourier de cada función $r_i(x, y)$ como $R_i(x, y)$, de acuerdo con el teorema de convolución se tiene que el espectro de la superposición de m rejillas está dada por (Goodman, 1996):

$$R(x, y) = R_1(x, y) * R_2(x, y) * \ldots * R_m(x, y), \tag{2}$$

donde $*$ denota la operación convolución en 2D. Así, aplicando la transformada inversa de Fourier a $R(x, y)$ se recupera la imagen de las rejillas superpuestas.

Como un primer acercamiento se pueden usar rejillas periódicas pues su espectro está compuesto por pulsos que se pueden caracterizar con 3 propiedades: índice en la matriz de la serie de Fourier, posición geométrica y amplitud (Figura 1). Dentro del estudio de estructuras periódicas, es de esperar encontrar simetría en los espectros, por lo que

usualmente los pulsos con vector de frecuencia f estarán acompañados por pulsos gemelos con vector de frecuencia $-f$ (Figura 1).

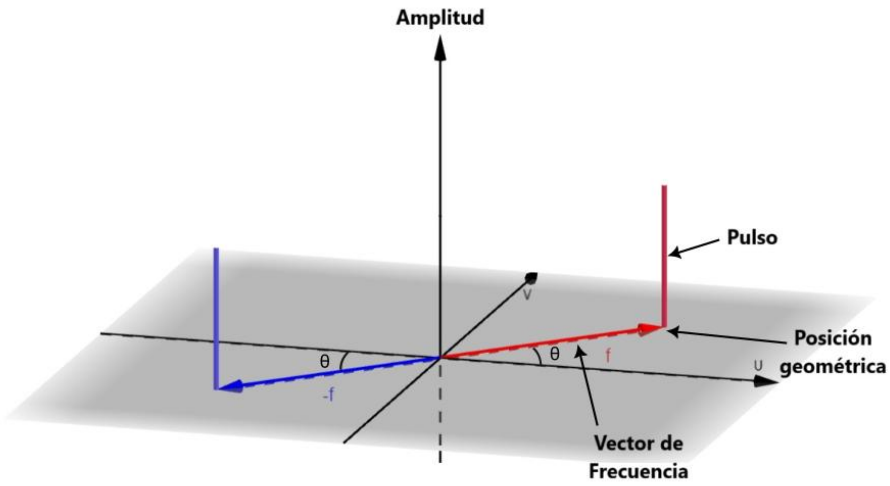

Figura 1: Amplitud y posición geométrica de pulsos en espectros 2D.

Finalmente en esta sección, en muchas ocasiones se requiere imprimir los patrones de franjas para realizar demostraciones de los efectos de Moiré. En este caso, como hay que superponer una de las rejillas sobre la otra, al menos una debe ser impresa sobre un medio transparente, por ejemplo, sobre hojas de acetato. La rejilla impresa en el medio transparente va en cima del otro, que puede ser impreso en papel. Esto permite ver la superposición de las dos rejillas claramente.

2. Rejillas cosinusoidales

Una rejilla con perfil de intensidad cosinusoidal es un buen punto de inicio, pues cumple con estar formada por valores entre 0 y 1, así considérese la siguiente función de reflectancia:

$$r(x,y) = Acos(2\pi f[xcos\theta + ysin\theta]) + DC, \quad (3)$$

donde A denota la amplitud de la función, f es la frecuencia de ciclos por unidad de distancia, θ es el ángulo que esté rotada la rejilla y DC representa la amplitud del pulso DC (esto es una intensidad constante sobre toda la imagen, que se convierte en un pulso

centrado en el origen de la transformada de Fourier). Así entonces, una rejilla no rotada, $\theta = 0$, de amplitud $A = 1/2$, frecuencia $f = 6$ ciclos por 500 pixeles, y $DC = 1/2$, se vería como se muestra en la Figura 2 (Goodman, 1996). Una rejilla de los mismos parámetros, pero rotada $\theta = \pi/4$, se muestra en la Figura 3.

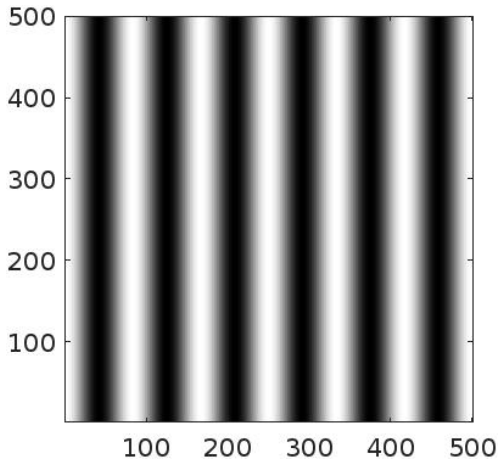

Figura 2: Rejilla cosinusoidal con $\theta = 0$, $A = 1/2$, $f_1 = 6$ ciclos por 500 pixeles, y $DC = 1/2$.

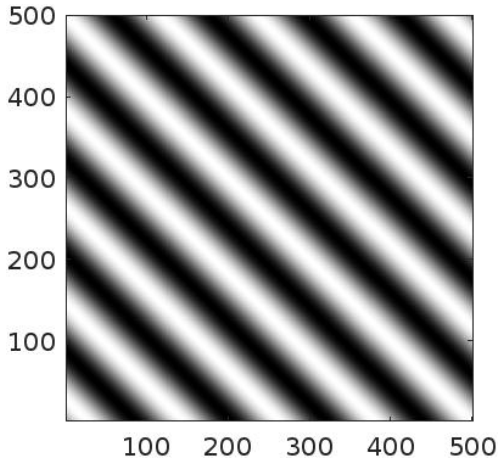

Figura 3: Rejilla cosinusoidal con $\theta = \pi/4$, $A = 1/2$, $f_2 = 6$ ciclos por 500 pixeles, y $DC = 1/2$.

Por otro lado, la transformada de Fourier de las rejillas estará compuesta por 3 pulsos, uno correspondiente a la frecuencia de las franjas, su gemelo y el impulso DC. Nótese

que en las transformadas de Fourier numéricas un “pulso” corresponde a un valor significativo en un solo píxel, rodeado por valores cercano de cero (no son ceros por el error numérico en el cálculo) en los pixeles vecinos. Se muestran los pulsos teóricos esperados de las dos rejillas en las Figuras 4 y 5.

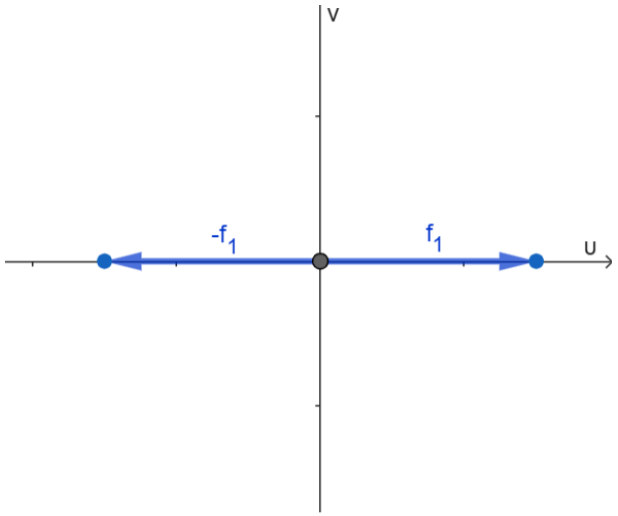

Figura 4: Espectro teórico de una rejilla cosinusoidal no rotada.

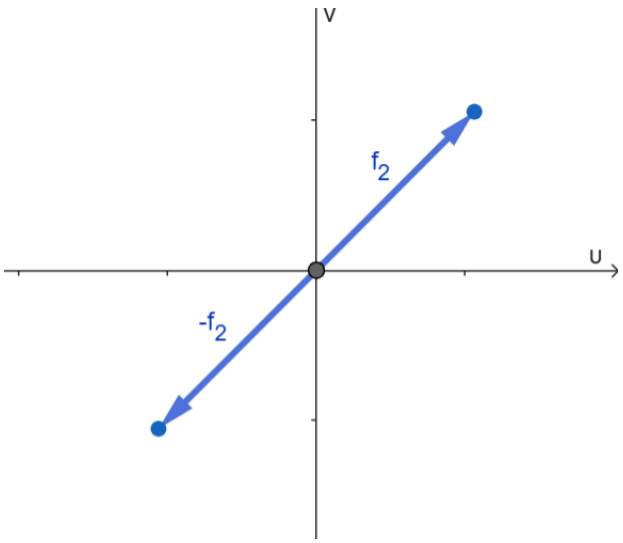

Figura 5: Espectro teórico de una rejilla cosinusoidal rotada $\theta = \pi/4$.

Se espera que la superposición de las rejillas tenga el espectro similar a lo mostrado en la Figura 6 (Amidror, 2009).

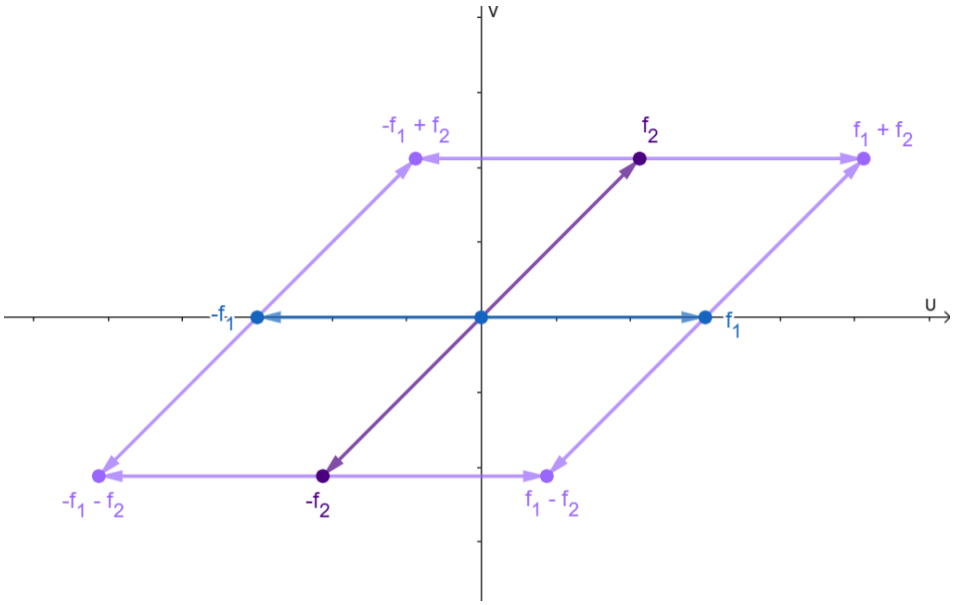

Figura 6: Espectro teórico (Ecuación 2) de dos rejillas cosinusoidales relativamente rotadas y superpuestas.

El espectro anterior está compuesto por los pulsos de las dos rejillas originales, cuya localización geométrica no se ve afectada pero sí su amplitud; además aparecen dos pares de nuevos pulsos que se caracterizan por ser la suma vectorial y la diferencia vectorial de los pulsos originales, esto debido a la operación que describe esta superposición es la convolución. Ahora bien, solo las amplitudes de las rejillas originales y de la rejilla superpuesta se verían como se muestran en la Figura 7 *(Amidror, 2009)*, en donde se puede ver cómo se hacen más frecuencias espaciales en la imagen superpuesta.

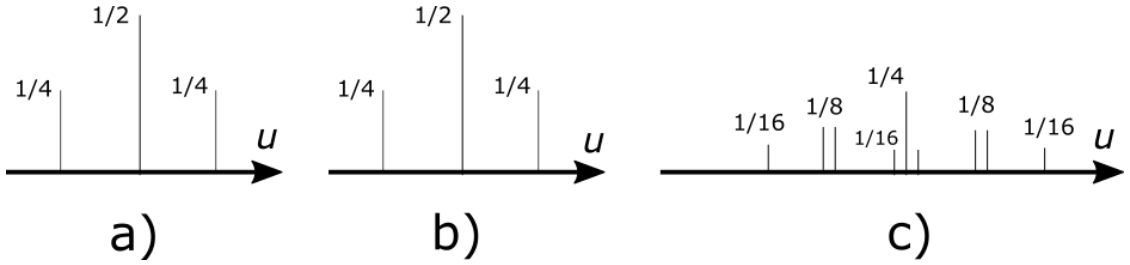

Figura 7: Amplitudes de los pulsos de las rejillas, a) rejilla no rotada; b) rejilla rotada; c) rejillas superpuestas. Figura tomada de (Amidror, 2009)

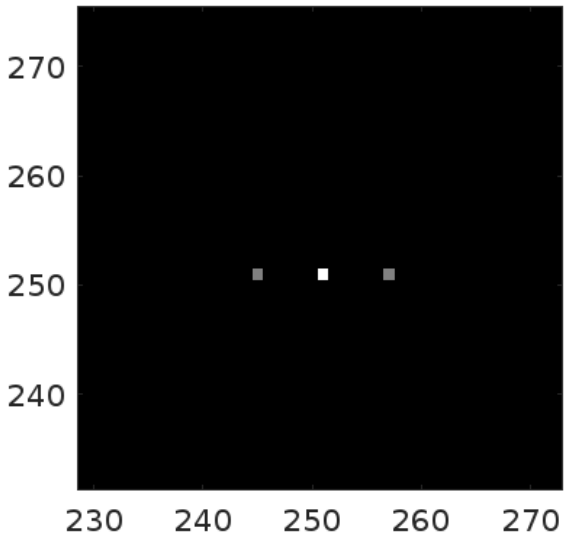

Figura 8: Espectro de la rejilla cosinusoidal no rotada en Matlab (Ampliación).

Para estudiar y producir los patrones de Moiré se utilizó el software *Matlab*, así que a lo largo del trabajo se presentarán los resultados obtenidos con dicho software. Con *Matlab* se obtuvieron los espectros de las rejillas de las Figuras 2 y 3, y se presentan en las Figuras 8 y 9. Este software produce la transformada de Fourier digital de señales limitados en tamaño o finitos. Esto es, se analiza señales con un número finito de pixeles. Las conclusiones utilizando la transformada de Fourier infinita arriba son válidas para los resultados numéricos considerando que un “pulso” es un píxel con un valor significativo rodeado con pixeles con valores casi cero.

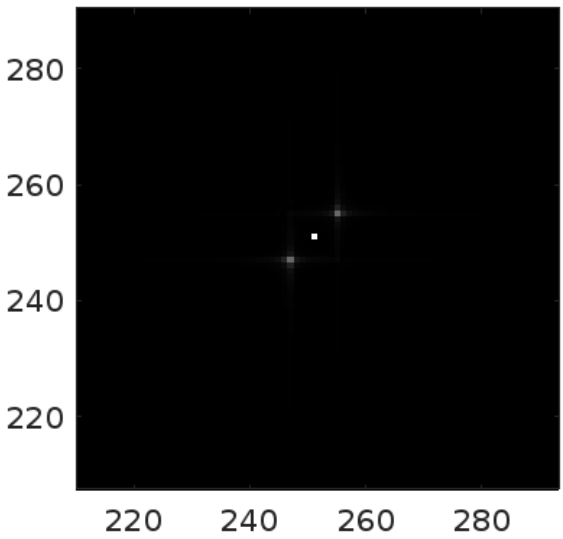

Figura 9: Espectro de la rejilla cosinusoidal rotada en Matlab (Ampliación).

El espectro de la convolución de los espectros de las Figuras 8 y 9 se muestra en la Figura 10. En la Figura 10 se puede observar un paralelogramo similar al de la Figura 6, como se esperaba.

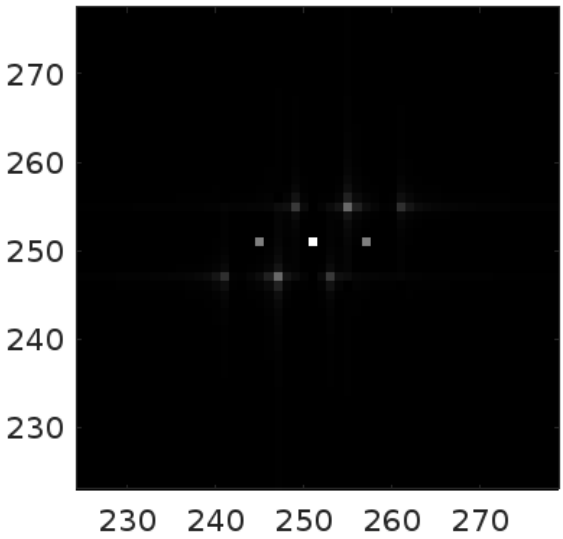

Figura 10: Convolución de los espectros de las rejillas rotada y no rotada en Matlab (Ampliación).

El siguiente paso consiste en aplicar la función de Transformada inversa de Fourier a la convolución obtenida, mostrado en la Figura 11, pues así se espera obtener la imagen de las rejillas superpuestas, también notemos que la imagen obtenida mediante este método es igual a la que se obtendría multiplicando directamente las matrices de las rejillas (mostrado en la Figura 12).

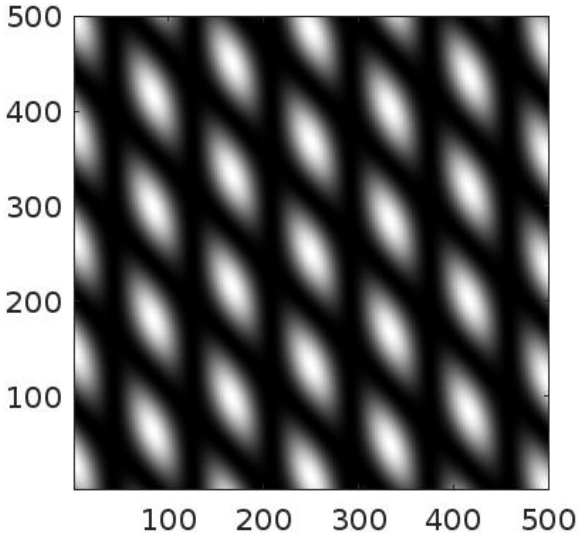

Figura 11: Transformada inversa de Fourier de la Figura 10 que da como resultado las rejillas superpuestas.

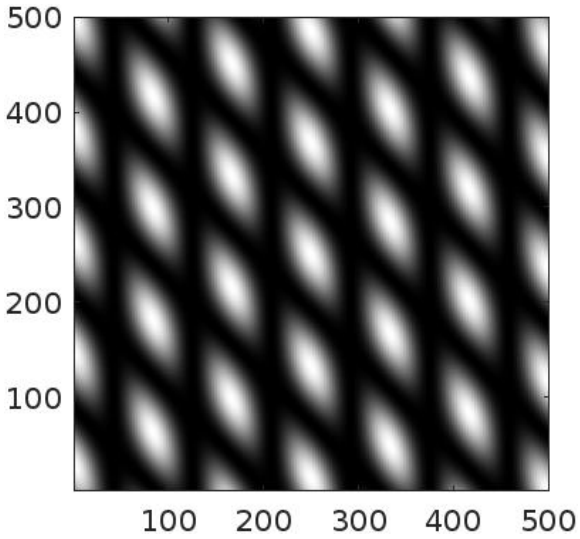

Figura 12: Multiplicación directa de ambas rejillas (Figuras 2 y 3).

Se puede apreciar la formación de nuevos patrones en forma de rombos en estas últimas imágenes, su aparición se debe a los pares de frecuencias producidos durante la convolución.

Para tener una mejor vista de los patrones producidos, una opción sería aplicar un filtrado a la imagen para sólo dejar las frecuencias de interés. Por ejemplo, para enfatizar la contribución de las nuevas frecuencias en la convolución de la Figura 10, y eliminar las frecuencias de las rejillas originales, se establece una amplitud crítica $A_{crítica}$ y se elimina los frecuencias en el espectro que tengan una amplitud mayor a esta amplitud crítica, y se puede hacer un filtrado de la rejilla resultante. Como ejemplo se hizo un filtro con $A_{\text{crítica}} = 0.1$ y como resultado tenemos el espectro mostrado en la Figura 13. Nótese que, para asegurar que las amplitudes en la imagen resultante son mayores a 1, no se filtra el término de la frecuencia 0, que es el término DC. Se puede apreciar que algunas frecuencias del paralelogramo original (Figura 10), ya no aparecen en la Figura 13, esto debido al criterio aplicado. Al aplicar la Transformada inversa de Fourier se recupera la rejilla mostrada en la Figura 14. La Figura 14 acentúa mejor las figuras nuevas repetidas del patrón de Moiré.

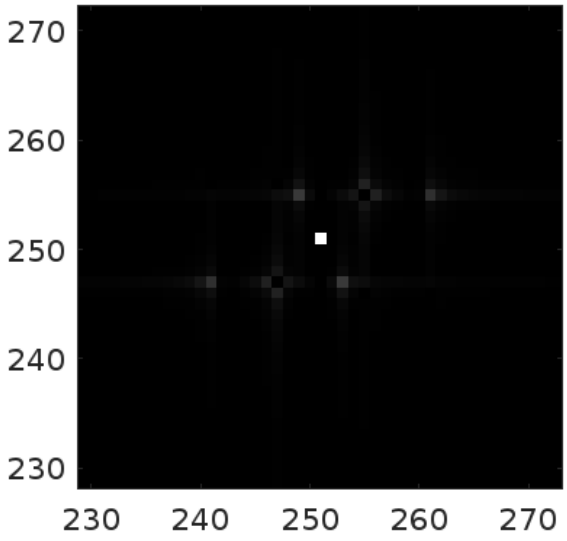

Figura 13: Espectro de la Figura 10 filtrado con amplitud crítica $A_{\text{crítica}} = 0.1$.

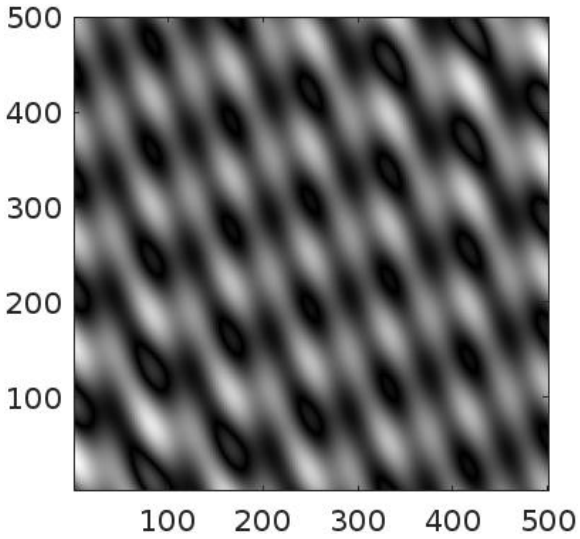

Figura 14: Transformada inversa de Fourier del espectro filtrado de la Figura 13.

2.1 Código *MatLab* para Rejillas Cosinusoidales

```
% REJILLAS COSINUSOIDALES
%
%-----------Valores iniciales-----------------------
amp = 1/2;          % amplitud
f1 = 3;             % frecuencia para rejilla 1
f2 = 3;             % frecuencia para rejilla 2
theta1 = 0.0;       % angulo para rejilla 1
theta2 = pi/4;      % angulo para rejilla 2
DC = 1/2;           % DC
xpx=500;            % resolucion de la rejilla en x
ypx=xpx;            % resolucion de la rejilla en y

%
% Generamos dos rejillas, una no rotada y la otra con rotacion
r1=cosgrid(amp,f1,theta1,DC,xpx,ypx);
r2=cosgrid(amp,f2,theta2,DC,xpx,ypx);

% Pedimos ver las graficas de ambas
figure
showImage(r1)
figure
showImage(r2)

% Obtenemos las transformadas de las rejillas
```

```
T_r1=fft2(r1)/(xpx*ypx);    % Transformada rejilla 1
T_r2=fft2(r2)/(xpx*ypx);    % Transformada rejilla 2
T_s1=fftshift(T_r1);        % Desplazamiento de Transformada 1 al centro
T_s2=fftshift(T_r2);        % Desplazamiento de Transformada 2 al centro

% Pedimos las gráficas de las amplitudes de las transformadas
T_a1=abs(T_s1);             % Amplitud de la Transformada 1
T_a2=abs(T_s2);              % Amplitud de la Transformada 2
figure
showImage(T_a1)
figure
showImage(T_a2)

% Convolucionamos las transformadas de las rejillas, notemos que no se usa
% la amplitud de la transformada si no el paso anterior, donde está toda la
% información.
% La funcion de Matlab conv2 no sirve para otras superposiciones
% Debido a la forma de parchar las matrices, nos conviene usar
% la funcion convolve2 que se encuentra en el repositorio de Matlab

Conv = convolve2(T_s1,T_s2,'circular'); % Convolucion de Transformadas


% Pedimos la grafica de la convolución
Conv_a = abs(Conv);             % Amplitud de la Convolucion
figure
showImage(Conv_a)

% Para mostrar el resultado de superponer las rejillas, aplicamos la
% Transformada inversa de Fourier y graficamos.

TI_r1r2=abs(ifft2(Conv));       % Transformada inversa de Fourier de la Convolucion
figure
showImage(TI_r1r2)

%Comparamos con la superposicion que resulta de multiplicar las matrices
figure
showImage(r1.*r2)

%-------OPCIONAL 1: FILTRADO-------------
% Podemos filtrar frecuencias de la convolución para solo quedarnos con las
% frecuencias que más influyen en los patrones.
% La función filtro requiere de entradas: una amplitud crítica (a_cr),
% la matriz de la transformada de la primera rejilla (mT1), la matriz de
% la transformada de la segunda rejilla (mT2) y devuelve la matriz del
% espectro filtrado.


Conv_filtrada=filtro(0.1,T_s1,T_s2);

% Pedimos la imagen del espectro con el filtro aplicado
figure
showImage(abs(Conv_filtrada))
```

```
% Pedimos la imagen de la Transformada inversa de Fourier del espectro filtrado.
figure
showImage(abs(ifft2(Conv_filtrada)))

%{
%-----------------GIF Cos-wave-----------------------
% Este codigo hace un gif de las rejillas superpuestas y cambia el angulo
% de la segunda rejilla. Definimos la cantidad de pasos que tendrá la
% animación en tstep.
tstep=50;

for i= 0:tstep
    theta2=(i*pi/tstep);  % angulo que varia en cada paso
    r1p=cosgrid(amp,f1,theta1,DC,xpx,ypx); % rejilla 1
    r2p=cosgrid(amp,f2,theta2,DC,xpx,ypx); % rejilla 2

   showImage(r1p.*r2p)

   frame =getframe(gcf);
   img =  frame2im(frame);
   [img,cmap] = rgb2ind(img,256);

   if i == 0
       imwrite(img,cmap,'Moire_cos_t+.gif','gif','DelayTime',0.1,'LoopCount',Inf);
   else
       imwrite(img,cmap,'Moire_cos_t+.gif','gif','WriteMode','append');
   end
end
%}

function [r] = cosgrid(a,f,t,dc,xpx,ypx)
% Función que hace rejillas cosinusoidales de tamaño xpix en el eje x, y
% ypix en el eje y.
% a es la amplitud
% f es la frecuencia de la rejilla
% t es el ángulo de la rejilla
% dc es el impulso DC

% Primero definimos la matriz donde estará almacenada la info de la rejilla
r=zeros(ypx,xpx);
% Generamos los valores de la rejilla
    for i = 1:ypx
        for j = 1:xpx
            r(i,j)=a*cos(2*pi*f*((-1+(2*j/xpx))*cos(t)+(-1+(2*i/ypx))*sin(t)))+dc;
        end
    end
end

function [Conv_F] = filtro(a_cr,mT1,mT2)
% Función para hacer filtrado, sólo deja las amplitudes por debajo de una
% amplitud crítica, se filtran desde las transformadas originales
% necesita la amplitud crítica (a_cr) y las transformadas de las rejillas
% originales (mt1, mT2)
[ypx1,xpx1]=size(mT1);
```

```
[ypx2,xpx2]=size(mT2);
mConv = convolve2(mT1,mT2,'circular');

% Más adelante necesitaremos reincorporar el valor maximo de la convolución
% de espectros, por lo que lo guardamos.
mConv_a=abs(mConv);
adc=max(max(mConv_a));

% Para comparar con la amplitud crítica, obtenemos las amplitudes de las
% transformadas
amT1=abs(mT1); % Amplitud transformada 1
amT2=abs(mT2); % Amplitud transformada 2

T_filt1=ones(ypx1,xpx1);     % Definimos matrices de unos para hacer los filtros
T_filt2=ones(ypx2,xpx2);
for i=1:ypx1             % Eliminamos las frecuencias de las rejillas base
   for j=1:xpx1
       if amT1(i,j)>a_cr
           T_filt1(i,j)=0;
       end
   end
end
for i=1:ypx2
    for j=1:xpx2
       if amT2(i,j)>a_cr
           T_filt2(i,j)=0;
       end
   end
end
Conv_F = mConv.*T_filt1.*T_filt2; % Filtramos la convolucion de imagenes
[ypxc,xpxc]=size(Conv_F);
Conv_F(ceil((ypxc/2)+1),ceil((xpxc/2)+1))=adc; % Reincorporamos la amplitud DC
end

function showImage(img,x,y)
   if ~exist('x','var')
       x = 1:size(img,2);
   end
   if ndims(x)==2
       x = x(1,:);
   end
   if ~exist('y','var')
       y = 1:size(img,1);
   end
   if ndims(y)==2
       y = y(:,1)';
   end

   imagesc(x,y,img)
    axis equal
    axis tight
   colormap(gray);
   set(gca,'YDir','normal')
end
```

3. Rejillas cuadradas

Otro tipo de rejillas que se adaptan a las hipótesis de este trabajo son las rejillas cuadradas, las cuales son similares a las rejillas cosinusoidales que se revisaron en las secciones anteriores, sin embargo la diferencia radica en que estas rejillas presentan bordes más definidos, lo que implica un espectro más complejo al momento de tomar la transformada de Fourier. En la Figura 15 se presenta una rejilla cuadrada sin rotación, y en la Figura 16 una rotada $\theta = \pi/4$.

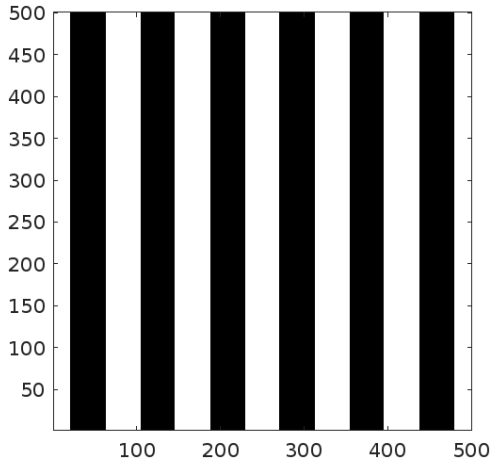

Figura 15: Rejilla cuadrada con $\theta = 0$, $A = 1/2$, $f = 6$ ciclos en 500 pixeles y $DC = 1/2$.

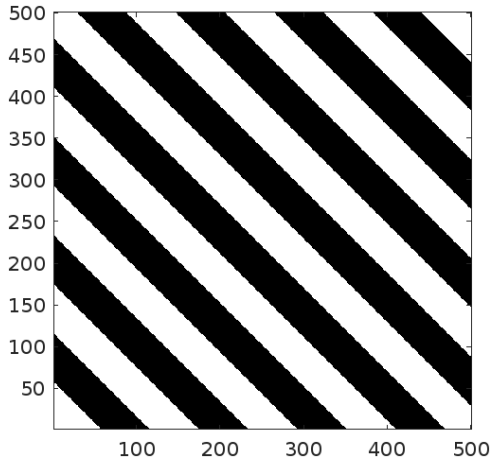

Figura 16: Rejilla cuadrada con $\theta = \pi/4$, $A = 1/2$, $f = 6$ ciclos en 500 pixeles y $DC = 1/2$.

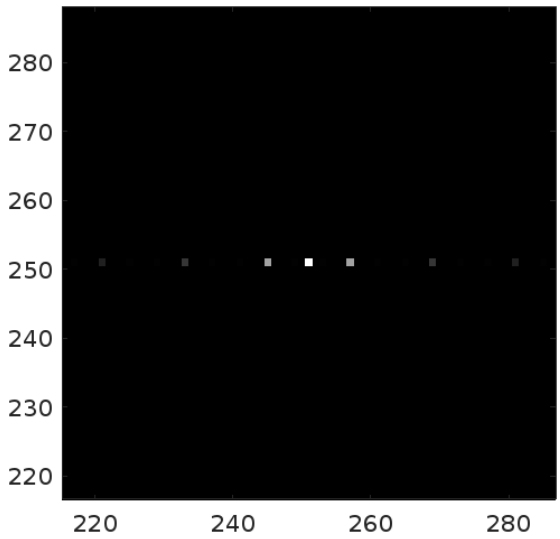

Figura 17: Espectro de la rejilla cuadrada no rotada en Matlab (Ampliación).

Las transformadas de Fourier digitales de las rejillas finitas individuales se presentan en Figuras 17 y 18, y la convolución de las dos en la Figura 19. Notemos que aparecen más amplitudes que en el caso cosinusoidal (Figuras 8 y 9), porque la serie de Fourier de una función rectángulo tiene más frecuencias.

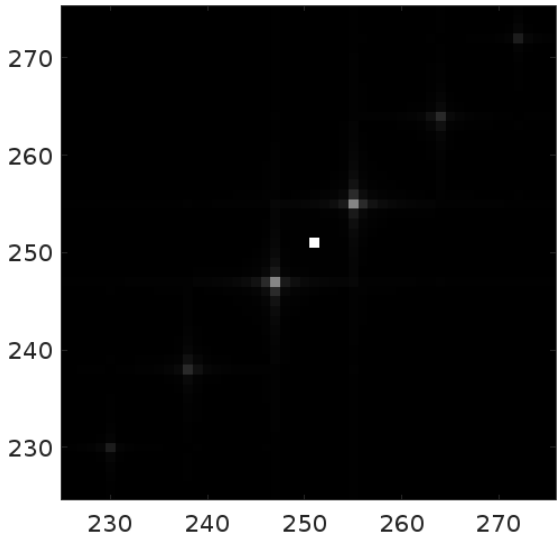

Figura 18: Espectro de la rejilla cuadrada rotada en Matlab (Ampliación).

Al aplicar la Transformada inversa de Fourier a la Figura 19, se recupera la siguiente imagen de las dos rejillas cuadradas superpuestas, mostrado en la Figura 20. Similar al caso cosinusoidal, se obtienen rombos que no estaban presentes en las rejillas originales.

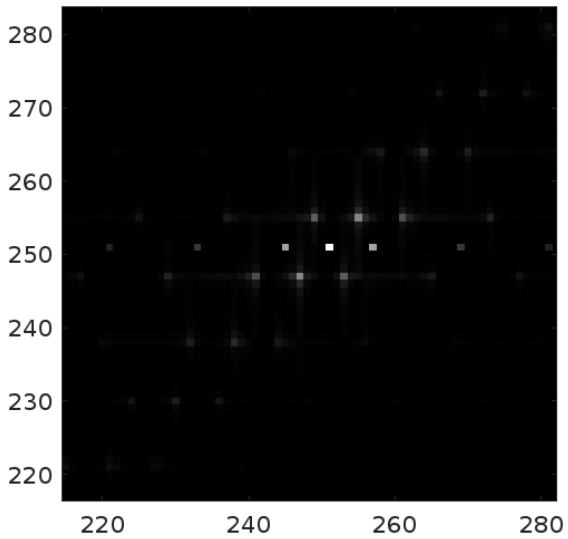

Figura 19: Convolución de los espectros de las rejillas cuadradas en Matlab (Ampliación).

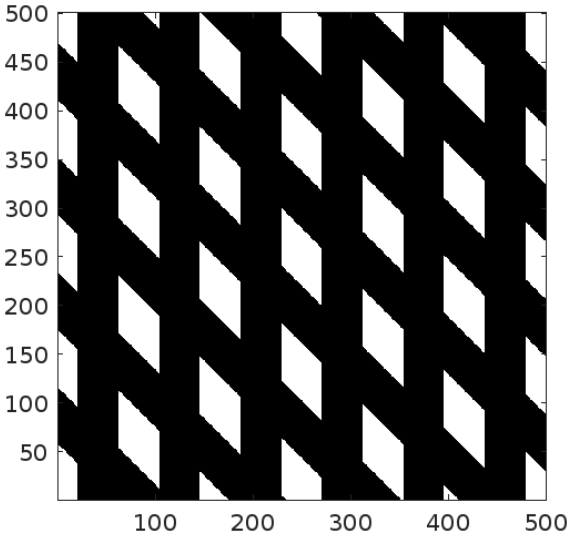

Figura 20: Transformada inversa de Fourier de la convolución de espectros de rejillas cuadradas.

De nuevo, se puede aplicar un filtro a la convolución de espectros para observar diferentes efectos, y en este caso, con la idea de quedarse sólo con amplitudes que jueguen un papel significativo en la formación de patrones, se probó un filtro con una amplitud crítica como se describió anteriormente, y que dejara sólo las amplitudes más cercanas al centro. Se hizo esto pensando en el llamado “círculo de visibilidad” (Amidror, 2009) que son las frecuencias que contribuyen más a la imagen detectada por el ojo, y por el hecho de que las amplitudes más grandes se encuentran más cercanas al centro. En la Figura 21 se presenta el espectro de la convolución de rejillas cuadradas filtrada con un valor $A_{\text{crítica}} = 0.02$ y radio de filtro de $r_{max} = 13$ pixeles. Otra vez se deja la frecuencia 0 para asegurar que la imagen es positiva.

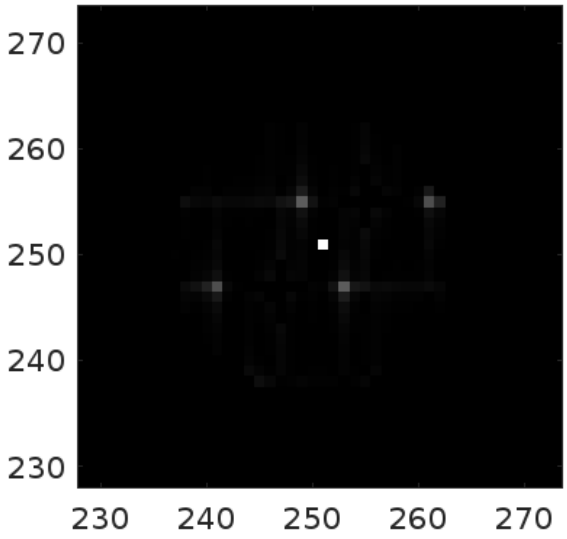

Figura 21: Espectro de convolución de rejillas cuadradas filtrada con $A_{\text{crítica}} = 0.02$ y $r_{max} = 13$ pixeles.

La imagen correspondiente se calcula con la Transformada inversa de Fourier de la Figura 21, y se muestra en la Figura 22. Se puede observar las formas de los rombos, con la misma orientación de los rombos de la Figura 20, pero con el doble de frecuencia, debido a las frecuencias quitados del espectro por el filtrado.

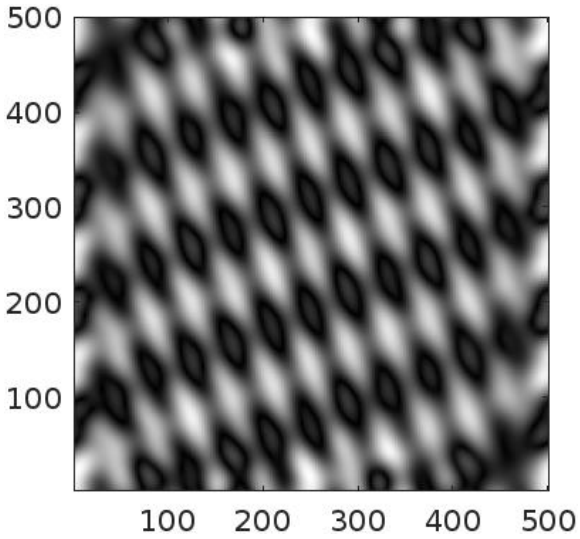

Figura 22: Transformada inversa de Fourier de espectro filtrado con $A_{\text{crítica}} = 0.02$ *y* $r_{max} = 13$ *pixeles.*

3.1 Código *MatLab* para Rejillas Cuadradas

```
% REJILLAS CUADRADAS

%
%-----------Valores iniciales-----------------------
amp = 1/2;          % amplitud
f1 = 3;             % frecuencia para rejilla 1
f2 = 3;             % frecuencia para rejilla 2
theta1 = 0.0;       % angulo para rejilla 1
theta2 = pi/4;      % angulo para rejilla 2
DC = 1/2;           % DC
xpx=500;            % resolucion de la rejilla en x
ypx=xpx;            % resolucion de la rejilla en y

%
% Generamos dos rejillas, una no rotada y la otra con rotacion
r1=sqrgrid(amp,f1,theta1,DC,xpx,ypx);
r2=sqrgrid(amp,f2,theta2,DC,xpx,ypx);

% Pedimos ver las graficas de ambas
figure
showImage(r1)
figure
showImage(r2)

% Obtenemos las transformadas de las rejillas
T_r1=fft2(r1)/(xpx*ypx);    % Transformada rejilla 1
```

```
T_r2=fft2(r2)/(xpx*ypx);    % Transformada rejilla 2
T_s1=fftshift(T_r1);        % Desplazamiento de Transformada 1 al centro
T_s2=fftshift(T_r2);        % Desplazamiento de Transformada 2 al centro

% Pedimos las gráficas de las amplitudes de las transformadas
T_a1=abs(T_s1);             % Amplitud de la Transformada 1
T_a2=abs(T_s2);              % Amplitud de la Transformada 2
figure
showImage(T_a1)
figure
showImage(T_a2)

% Convolucionamos las transformadas de las rejillas, notemos que no se usa
% la amplitud de la transformada si no el paso anterior, donde está toda la
% información.
% La funcion de Matlab conv2 no sirve para otras superposiciones
% Debido a la forma de parchar las matrices, nos conviene usar
% la funcion convolve2 que se encuentra en el repositorio de Matlab
Conv = convolve2(T_s1,T_s2,'circular'); % Convolucion de Transformadas

% Pedimos la grafica de la convolución
Conv_a = abs(Conv);             % Amplitud de la Convolucion
figure
showImage(Conv_a)

% Para mostrar el resultado de superponer las rejillas, aplicamos la
% Transformada inversa de Fourier y graficamos.
TI_r1r2=abs(ifft2(Conv));       % Transformada inversa de Fourier de la Convolucion
figure
showImage(TI_r1r2)

%-------OPCIONAL 1: FILTRADO-------------
% Podemos filtrar frecuencias de la convolución para solo quedarnos con las
% frecuencias que más influyen en los patrones.
% La función filtro requiere de entradas: una amplitud crítica (a_cr), un
% radio para eliminar TODAS las frecuencias fuera del círculo de dicho
% radio (rad), la matriz de la transformada de la primera rejilla (mT1),
% la matriz de la transformada de la segunda rejilla (mT2) y devuelve la
% matriz del espectro filtrado.
Conv_filtrada=filtrocir(0.02,13,T_s1,T_s2);

% Pedimos la imagen del espectro con el filtro aplicado
figure
showImage(abs(Conv_filtrada))

% Pedimos la imagen de la Transformada inversa de Fourier del espectro filtrado.
figure
showImage(abs(ifft2(Conv_filtrada)))
%}

%{
%-----------------OPCIONAL 2: GIF SQUARE-wave +ANGULO-----------------------
% Este codigo hace un gif de las rejillas superpuestas y cambia el angulo
% de la segunda rejilla. Definimos la cantidad de pasos que tendrá la
```

```
% animación en tstep.
tstep=50;
for i= 0:tstep
    theta2=(i*pi/tstep);  % angulo que varia en cada paso
    r1p=sqrgrid(amp,f1,theta1,DC,xpx,ypx); % rejilla 1
    r2p=sqrgrid(amp,f2,theta2,DC,xpx,ypx); % rejilla 2
   showImage(r1p.*r2p)

   frame =getframe(gcf);
   img =  frame2im(frame);
   [img,cmap] = rgb2ind(img,256);

   if i == 0
       imwrite(img,cmap,'Moire_sqr_t+.gif','gif','DelayTime',0.1,'LoopCount',Inf);
   else
       imwrite(img,cmap,'Moire_sqr_t+.gif','gif','WriteMode','append');
   end
end
%}

%{
%-----------------OPCIONAL 3: GIF SQUARE-wave +FRECUENCIA-----------------------
% Este codigo hace un gif de las rejillas superpuestas y cambia la
% frecuencia de la segunda rejilla. Definimos la cantidad de pasos que
% tendrá la animación en tstep. Para que la animación tenga más sentido
% se sugiere usar las dos rejillas no rotadas (checar valores iniciales)
tstep=15;
for i= 0:tstep
    f2=f2+(0.1)*i;  % frecuencia 2 que varia en cada paso
    r1p=sqrgrid(amp,f1,theta1,DC,xpx,ypx); % rejilla 1
    r2p=sqrgrid(amp,f2,theta2,DC,xpx,ypx); % rejilla 2
    showImage(r1p.*r2p)

   frame =getframe(gcf);
   img =  frame2im(frame);
   [img,cmap] = rgb2ind(img,256);

   if i == 0
       imwrite(img,cmap,'Moire_sqr_f+.gif','gif','DelayTime',0.1,'LoopCount',Inf);
   else
       imwrite(img,cmap,'Moire_sqr_f+.gif','gif','WriteMode','append');
   end
end
%}

%{
%-----------------OPCIONAL 4: GIF SQUARE-wave TRASLACION-----------------------
% Este codigo hace un gif de las rejillas superpuestas y cambia traslada
% la segunda rejilla. Definimos la cantidad de pasos que
% tendrá la animación en tstep. Para que la animación tenga más sentido
% se sugiere usar las dos rejillas no rotadas (checar valores iniciales)
tstep=20;
Xstep=2./xpx;
Ystep=2./ypx;
```

```
Xsup=1-Xstep;
Ysup=1-Ystep;
[X,Y]=meshgrid(-1:Xstep:Xsup,-1:Ystep:Ysup);

for i= 0:tstep
    r1p=amp*sign(cos(2*pi*f1*(X*cos(theta1)+Y*sin(theta1))))+ DC;
    r2p=amp*sign(cos(2*pi*(f2)*((X-0.02*i)*cos(theta2)+Y*sin(theta2))))+ DC;
    showImage(r1p.*r2p)

   frame =getframe(gcf);
   img =  frame2im(frame);
   [img,cmap] = rgb2ind(img,256);

   if i == 0
       imwrite(img,cmap,'Moire_sqr_x+.gif','gif','DelayTime',0.1,'LoopCount',Inf);
   else
       imwrite(img,cmap,'Moire_sqr_x+.gif','gif','WriteMode','append');
   end
end
%}
%

function [r] = sqrgrid(a,f,t,dc,xpx,ypx)
% Función que hace rejillas cosinusoidales de tamaño xpix en el eje x, y
% ypix en el eje y.
% a es la amplitud
% f es la frecuencia de la rejilla
% t es el ángulo de la rejilla
% dc es el impulso DC

% Primero definimos la matriz donde estará almacenada la info de la rejilla
r=zeros(ypx,xpx);

% Generamos los valores de la rejilla
    for i = 1:ypx
        for j = 1:xpx
            r(i,j)=a*sign(cos(2*pi*f*((-1+(2*j/xpx))*cos(t)+(-
1+(2*i/ypx))*sin(t))))+dc;
        end
    end
end

function [Conv_F] = filtrocir(a_cr,rad,mT1,mT2)
% Función para hacer filtrado, sólo deja las amplitudes por debajo de una
% amplitud crítica dentro del circulo de radio rad, se filtran desde las
% transformadas originales necesita la amplitud crítica (a_cr), el radio
% en pixeles del círculo de prefiltrado (rad) y las transformadas de las
% rejillas originales (mt1, mT2)
[ypx1,xpx1]=size(mT1);
[ypx2,xpx2]=size(mT2);
mConv = convolve2(mT1,mT2,'circular');

% Más adelante necesitaremos reincorporar el valor maximo de la convolución
% de espectros, por lo que lo guardamos.
```

```
mConv_a=abs(mConv);
adc=max(max(mConv_a));

% Para comparar con la amplitud crítica, obtenemos las amplitudes de las
% transformadas
amT1=abs(mT1); % Amplitud transformada 1
amT2=abs(mT2); % Amplitud transformada 2

% Definimos matrices de unos para hacer los filtros
T_filt1=ones(ypx1,xpx1);
T_filt2=ones(ypx2,xpx2);

% Prefiltramos la imagen despreciando TODAS las frecuencias fuera del
% circulo de radio rad.
for i=1:ypx1
    for j=1:xpx1
        if (j-ceil(xpx1/2)).^2+(i-ceil(ypx1/2)).^2>rad^2
            T_filt1(i,j)=0;
        end
    end
end

for i=1:ypx2
    for j=1:xpx2
        if (j-ceil(xpx2/2)).^2+(i-ceil(ypx2/2)).^2>rad^2
            T_filt2(i,j)=0;
        end
    end
end
for i=1:ypx1             % Eliminamos las frecuencias de las rejillas base
   for j=1:xpx1
       if amT1(i,j)>a_cr
           T_filt1(i,j)=0;
       end
   end
end
for i=1:ypx2
    for j=1:xpx2
       if amT2(i,j)>a_cr
           T_filt2(i,j)=0;
       end
   end
end
Conv_F = mConv.*T_filt1.*T_filt2; % Filtramos la convolucion de imagenes
[ypxc,xpxc]=size(Conv_F);
Conv_F(ceil((ypxc/2)+1),ceil((xpxc/2)+1))=adc; % Reincorporamos la amplitud DC
end

function showImage(img,x,y)
   if ~exist('x','var')
       x = 1:size(img,2);
   end
   if ndims(x)==2
       x = x(1,:);
```

```
    end
    if ~exist('y','var')
        y = 1:size(img,1);
    end
    if ndims(y)==2
        y = y(:,1)';
    end

    imagesc(x,y,img)
     axis equal
     axis tight
    colormap(gray);
    set(gca,'YDir','normal')
end
```

4. Rejillas de puntos

La Figura 23 muestra un arreglo de puntos obtenido de: https://www.pngkey.com/detail/u2t4o0y3o0e6q8u2_points-pattern-puntos-patron-black-cute-sweet-line/#google_vignette.

De forma similar a las otras rejillas, la Figura 24 muestra una rotación de la rejilla de la Figura 23 por 10°. Se puede ver que la rotación provoca una pérdida de información cerca de los bordes, sin embargo, a continuación, se verá que no constituye un problema a nuestra forma de tratar las imágenes.

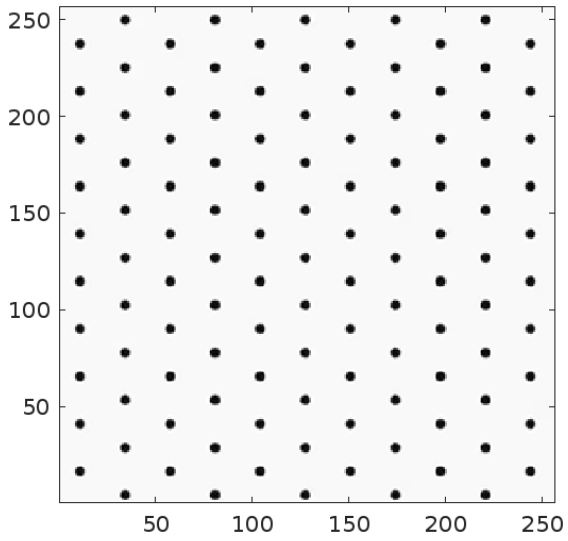

Figura 23: Rejilla de puntos

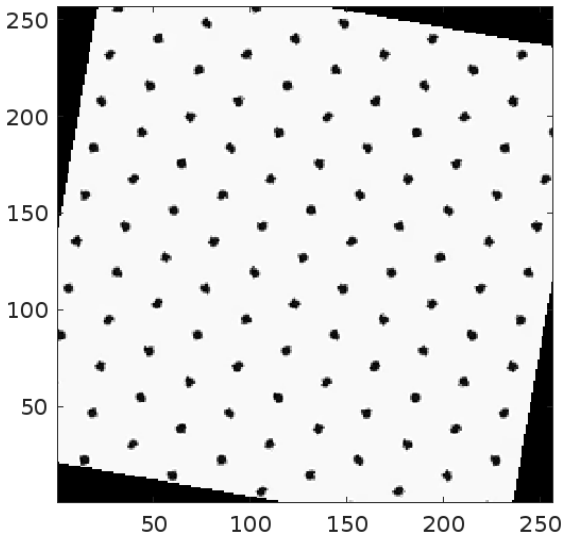

Figura 24: Rejilla de puntos rotada $\theta = 10°$

El resultado de la convolución de los espectros de muestra en la Figura 25. Nótese que las amplitudes de los espectros de las rejillas originales son muy pequeños y no logran apreciarse bien, por lo que se aplicó la función logaritmo a los datos para una mejor visualización.

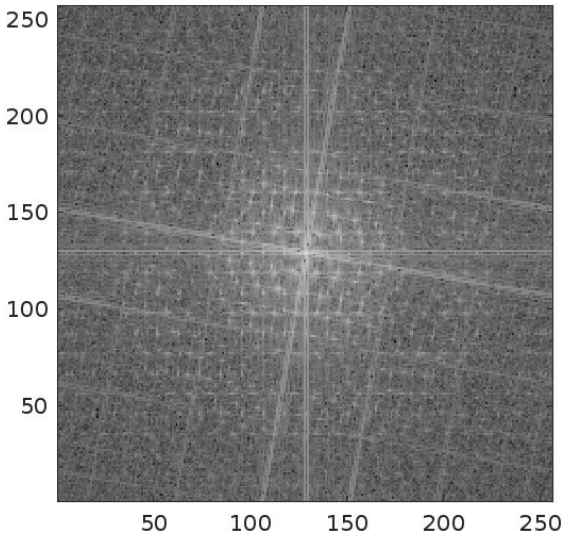

Figura 25: Convolución de espectros de rejillas de puntos en escala logarítmica.

Aplicando la transformada inversa de Fourier a la convolución de espectros se recupera la imagen de la Figura 26. En este caso se observa la formación de hexágonos que no estaban en las rejillas originales.

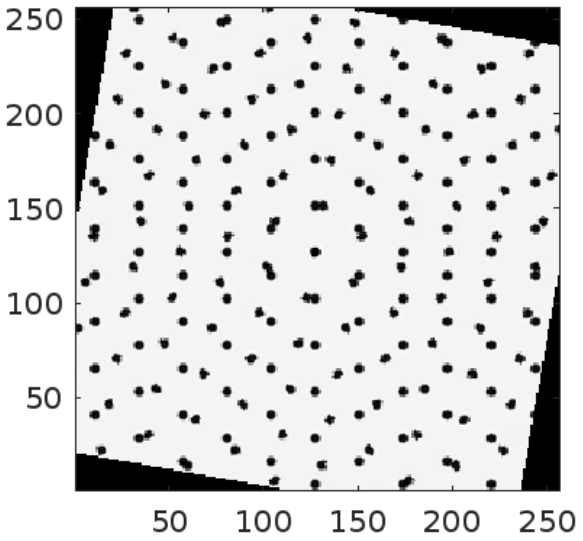

Figura 26: Transformada inversa de Fourier de la convolución de espectros de las rejillas de puntos (Figura 25).

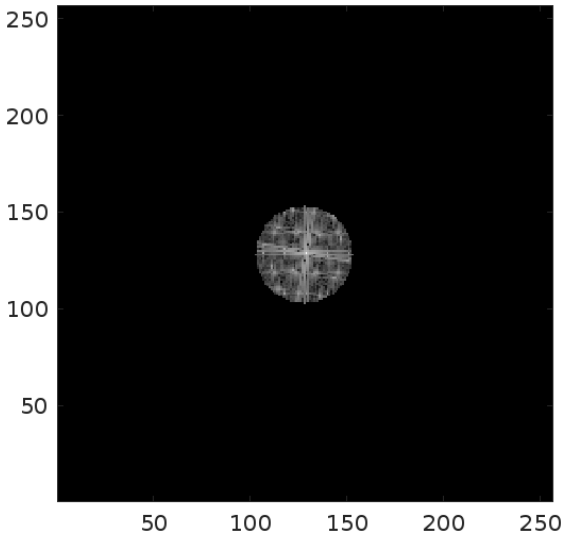

Figura 27: Convolución de espectro de rejillas filtrada con amplitud crítica $A_{\text{crítica}} = m \times 0.3$ y radio $r_{max} = 25$ pixeles.

Aplicando un filtro se pueden observar mejor los patrones de hexágonos. En la Figura 27 se presenta la convolución de espectros filtrada con una amplitud crítica de $A_{\text{crítica}} = m \times 0.3$ donde m simboliza la amplitud máxima de la convolución sin filtrar. Entonces, esta $A_{\text{crítica}}$ solo deja pasar aquellos valores por debajo del 30% de la máxima amplitud en la convolución original; por otro lado, para el radio se usó $r_{max} = 25$ pixeles.

Aplicando la transformada inversa de Fourier, se obtiene la imagen mostrada en la Figura 28. Se puede notar de la Figura 28 que se ha enfatizado las formas hexagonales en el patrón.

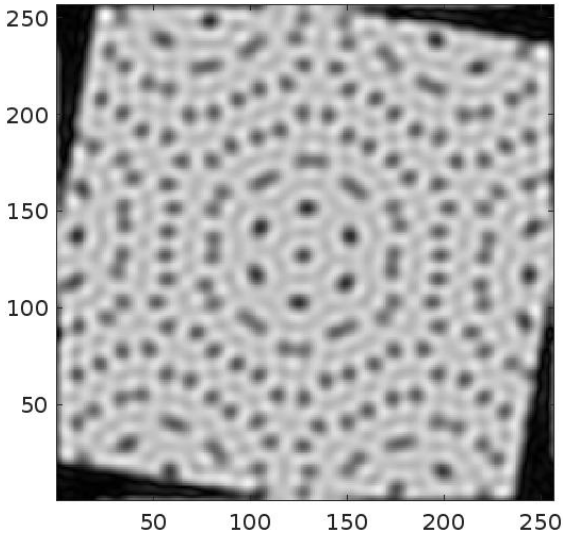

Figura 28: Transformada inversa de Fourier de la convolución filtrada de espectros de rejillas de puntos.

Otro aspecto importante de esta rejilla es su evolución a través del cambio de ángulo de rotación, en las Figuras 29 a 36, se presentan la superposición a varios ángulos de las rejillas. Se puede notar como el número y distribución de los hexágonos va cambiando al cambiar el ángulo de rotación de la segunda imagen.

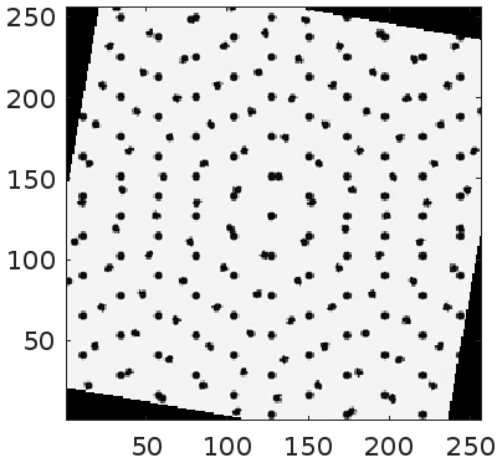

Figura 29: $\theta = 10°$

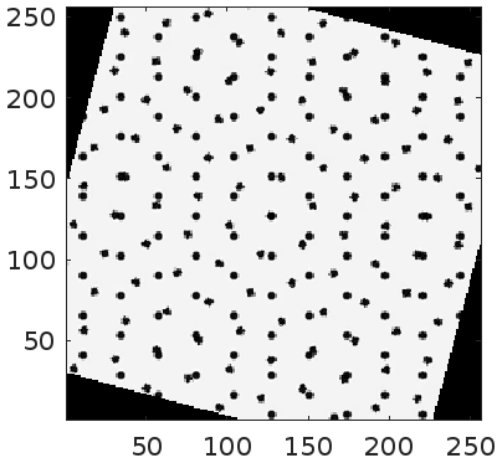

Figura 30: $\theta = 15°$

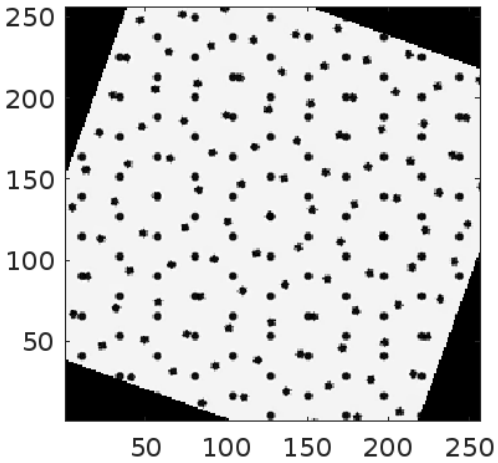

Figura 31: $\theta = 20°$

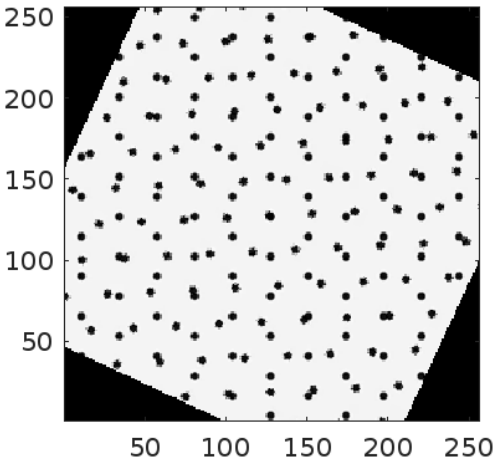

Figura 32: $\theta = 25°$

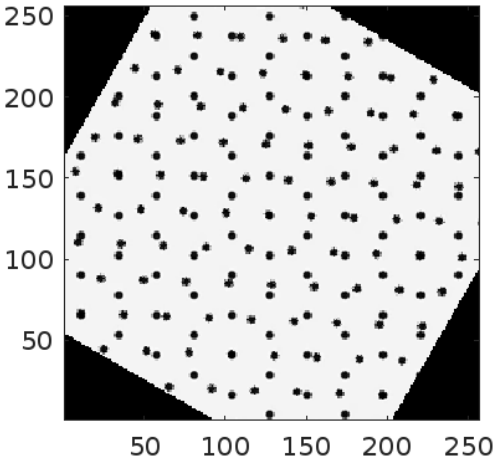

Figura 33: $\theta = 30°$

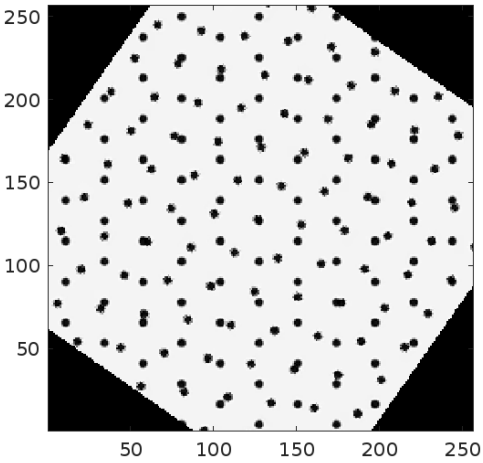

Figura 34: $\theta = 35°$

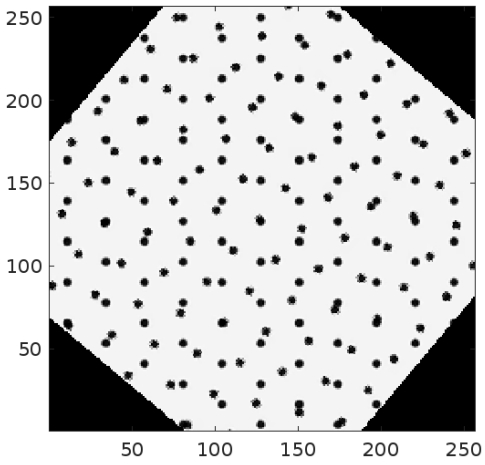

Figura 35: $\theta = 40°$

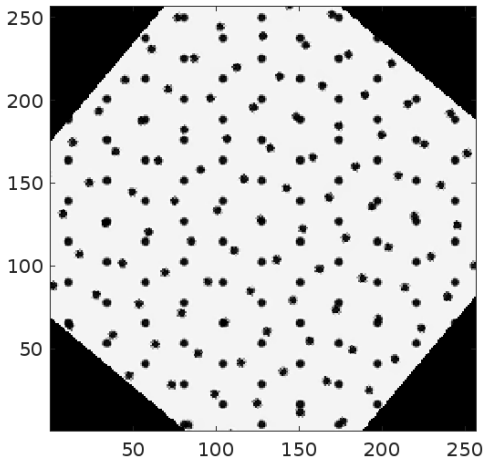

Figura 36: $\theta = 45°$

4.1 Código *MatLab* para Rejillas de puntos

```
%----------------- Dos patrones de puntos --------------------------------
% Digitalizamos el patron a utilizar:
A=imread('patron-puntos.png');

% Lo pasamos a un formato a blanco y negro
A=double(rgb2gray(A));

% Y la dejamos en un tamaño estándar
A=imcrop(A,[1 1 255 255]);
[Ay,Ax]=size(A);

% Pedimos que nos muestre la primera rejilla
figure
showImage(A)

% Para crear la segunda rejilla, rotamos la primera rejilla, la funcion esta
% en grados, esto lo definimos en ang
ang=10; % Grados por girar
B=imrotate(A,ang,'crop');
[By,Bx]=size(B);

% Y podemos pedir la vista de esta segunda rejilla
figure
showImage(B)

% Obtenemos las transformadas de las rejillas
```

```
T_A=fft2(A)/(Ax*Ay);    % Transformada rejilla 1
T_B=fft2(B)/(Bx*By);    % Transformada rejilla 2
T_sA=fftshift(T_A);        % Desplazamiento de Transformada 1 al centro
T_sB=fftshift(T_B);        % Desplazamiento de Transformada 2 al centro

% Convolucionamos las transformadas de las rejillas.
% La funcion de Matlab conv2 no sirve para otras superposiciones
% Debido a la forma de parchar las matrices, nos conviene usar
% la funcion convolve2 que se encuentra en el repositorio de Matlab
Conv = convolve2(T_sA,T_sB,'circular'); % Convolucion de Transformadas
adc=max(max(abs(Conv))); % Nos servirá de referencia para saber el tamaño
                         % amplitudes con las que trabajamos.

% Para mostrar el resultado de superponer las rejillas, aplicamos la
% Transformada inversa de Fourier y graficamos.
TI_AB=abs(ifft2(Conv));       % Transformada inversa de Fourier de la Convolucion
figure
showImage(TI_AB)

%{
%-----OPCIONAL 1: GRAFICAS ESPECTROS
T_aA=abs(T_sA);             % Amplitud de la Transformada 1
T_aB=abs(T_sB);             % Amplitud de la Transformada 2
Conv_a = abs(Conv);         % Amplitud de la Convolucion
figure
showImage(log(T_aA))
figure
showImage(log(T_aB))
figure
showImage(log(Conv_a))
%}

%{
%-------OPCIONAL 2: FILTRADO-------------
% Podemos filtrar frecuencias de la convolución para solo quedarnos con las
% frecuencias que más influyen en los patrones.
% La función filtro requiere de entradas: una amplitud crítica (a_cr), un
% radio para eliminar TODAS las frecuencias fuera del círculo de dicho
% radio (rad), la matriz de la transformada de la primera rejilla (mT1),
% la matriz de la transformada de la segunda rejilla (mT2) y devuelve la
% matriz del espectro filtrado.
Conv_filtrada=filtrocir(adc*0.3,25,T_sA,T_sB);
% Pedimos la imagen del espectro con el filtro aplicado, dado que las
% amplitudes son pequeñas, conviene visualizar el espectro en escala
% logaritmica
figure
showImage(log(abs(Conv_filtrada)))
% Pedimos la imagen de la Transformada inversa de Fourier del espectro filtrado.
figure
showImage(abs(ifft2(Conv_filtrada)))
%}

%{
%-----------------OPCIONAL 3: GIF PUNTOS +ANGULO-----------------------
```

```
% Este codigo hace un gif de las rejillas superpuestas y cambia el angulo
% de la segunda rejilla. Definimos la cantidad de pasos que tendrá la
% animación en tstep.
tstep=10;
for i= 0:tstep
    ang=5*i; % Grados por girar
    B=imrotate(A,ang,'crop');
    showImage(A.*B)

   frame =getframe(gcf);
   img =  frame2im(frame);
   [img,cmap] = rgb2ind(img,256);

   if i == 0
       imwrite(img,cmap,'Moire_dot_t+.gif','gif','DelayTime',0.1,'LoopCount',Inf);
   else
       imwrite(img,cmap,'Moire_dot_t+.gif','gif','WriteMode','append');
   end
end
%}



function [Conv_F] = filtrocir(a_cr,rad,mT1,mT2)
% Función para hacer filtrado, sólo deja las amplitudes por debajo de una
% amplitud crítica dentro del circulo de radio rad, se filtran desde las
% transformadas originales necesita la amplitud crítica (a_cr), el radio
% en pixeles del círculo de prefiltrado (rad) y las transformadas de las
% rejillas originales (mt1, mT2)
[ypx1,xpx1]=size(mT1);
[ypx2,xpx2]=size(mT2);
mConv = convolve2(mT1,mT2,'circular');

% Más adelante necesitaremos reincorporar el valor maximo de la convolución
% de espectros, por lo que lo guardamos.
mConv_a=abs(mConv);
adc=max(max(mConv_a));

% Para comparar con la amplitud crítica, obtenemos las amplitudes de las
% transformadas
amT1=abs(mT1); % Amplitud transformada 1
amT2=abs(mT2); % Amplitud transformada 2

% Definimos matrices de unos para hacer los filtros
T_filt1=ones(ypx1,xpx1);
T_filt2=ones(ypx2,xpx2);

% Prefiltramos la imagen despreciando TODAS las frecuencias fuera del
% circulo de radio rad.
for i=1:ypx1
    for j=1:xpx1
        if (j-ceil(xpx1/2)).^2+(i-ceil(ypx1/2)).^2>rad^2
            T_filt1(i,j)=0;
        end
```

```
    end
end

for i=1:ypx2
    for j=1:xpx2
        if (j-ceil(xpx2/2)).^2+(i-ceil(ypx2/2)).^2>rad^2
            T_filt2(i,j)=0;
        end
    end
end
for i=1:ypx1             % Eliminamos las frecuencias de las rejillas base
   for j=1:xpx1
       if amT1(i,j)>a_cr
           T_filt1(i,j)=0;
       end
   end
end
for i=1:ypx2
    for j=1:xpx2
       if amT2(i,j)>a_cr
           T_filt2(i,j)=0;
       end
   end
end

Conv_F = mConv.*T_filt1.*T_filt2; % Filtramos la convolucion de imagenes
[ypxc,xpxc]=size(Conv_F);
Conv_F(ceil((ypxc/2)+1),ceil((xpxc/2)+1))=adc; % Reincorporamos la amplitud maxima
end

function showImage(img,x,y)
   if ~exist('x','var')
       x = 1:size(img,2);
   end
   if ndims(x)==2
       x = x(1,:);
   end
   if ~exist('y','var')
       y = 1:size(img,1);
   end
   if ndims(y)==2
       y = y(:,1)';
   end

   imagesc(x,y,img)
    axis equal
    axis tight
   colormap(gray);
   set(gca,'YDir','normal')
end
```

5. Anillos concéntricos

Usando dos rejillas de anillos concéntricos de la misma forma mostrado en la Figura 37, y desplazando una con relación al otro en la dirección horizontal, se obtiene un patrón como la mostrado en la Figura 38. El desplazamiento para la Figura 38 fue de 20 pixeles. Se puede notar la creación de nuevas líneas radiales por la superposición.

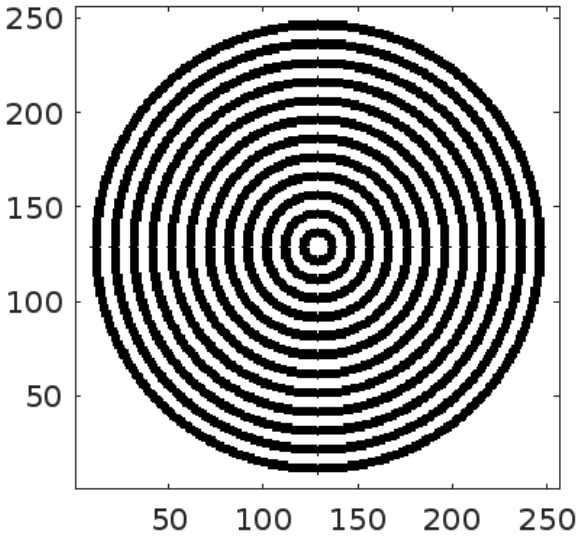

Figura 37: Rejilla de anillos concéntricos.

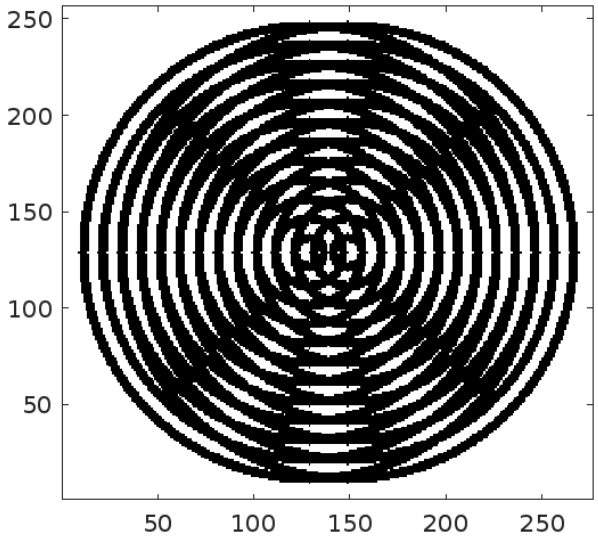

Figura 38: Superposición de anillos concéntricos con un desplazamiento lateral.

Para una mejor visualización, y recordando que se utilizan la transformada Fourier digital y señales finitas, se puede aplicar un filtro a la convolución de espectros. La Figura 39 muestra la convolución de los dos imágenes de anillos, con un desplazamiento entre ellos de 20 pixeles.

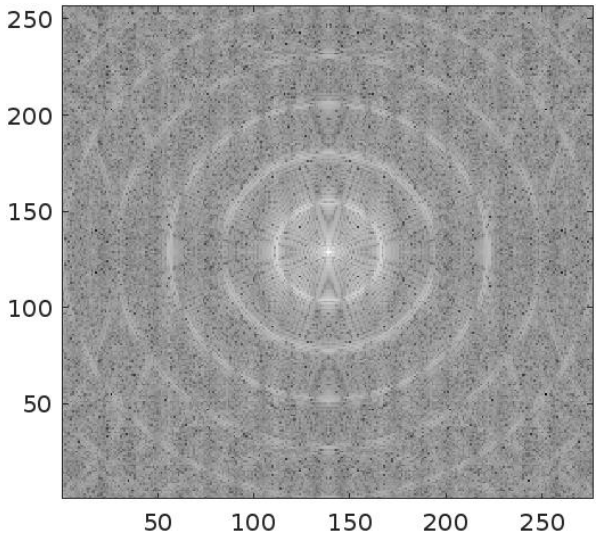

Figura 39: Convolución de espectros de anillos concéntricos en escala logarítmica.

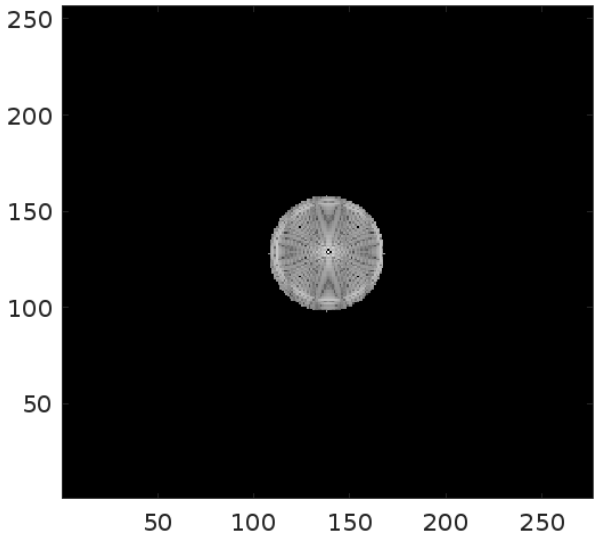

Figura 40: Convolución de espectros de anillos, filtrada con $A_{\text{crítica}} = m \times 0.1$ *y* $r_{max} = 30$ *pixeles en escala logarítmica.*

Para enfatizar los patrones nuevos en la superposición, se realizó un filtrado con $A_{\text{crítica}} = m \times 0.1$ y $r_{max} = 30$ pixeles, el espectro queda como se muestra en la Figura 40. Lo cual regresa la imagen de la Figura 41 al aplicar la transformada de Fourier inversa.

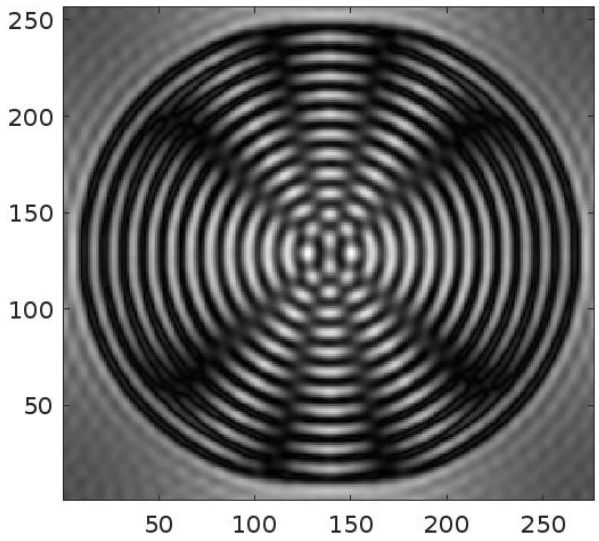

Figura 41: Transformada inversa de Fourier de la convolución filtrada de anillos.

Las Figuras 42 a 49 muestran la forma en que los patrones varían de acuerdo con la distancia que haya entre los centros. Se puede ver que a mayor separación de los dos imágenes de anillos, mayor número de líneas radiales aparecen.

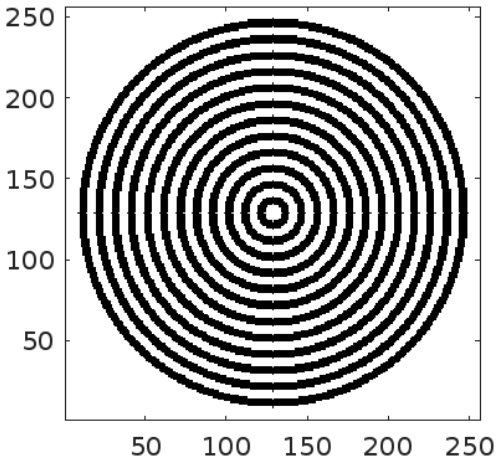

Figura 42: Desplazamiento 0 pixeles

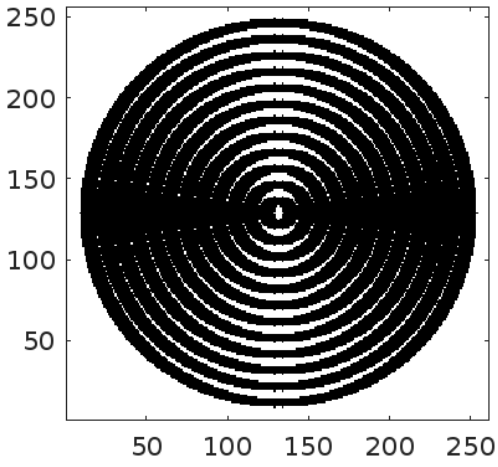

Figure 43: Desplazamiento 5 pixeles

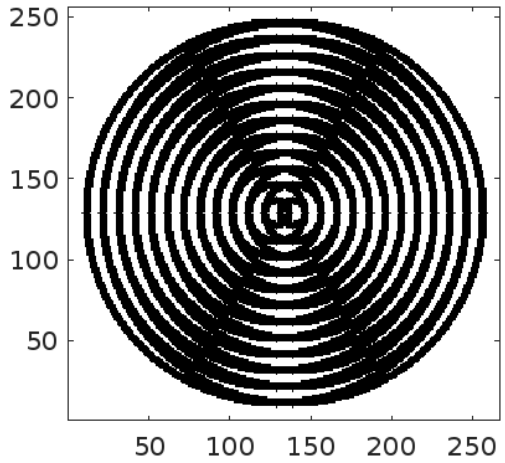

Figura 44: Desplazamiento 10 pixeles

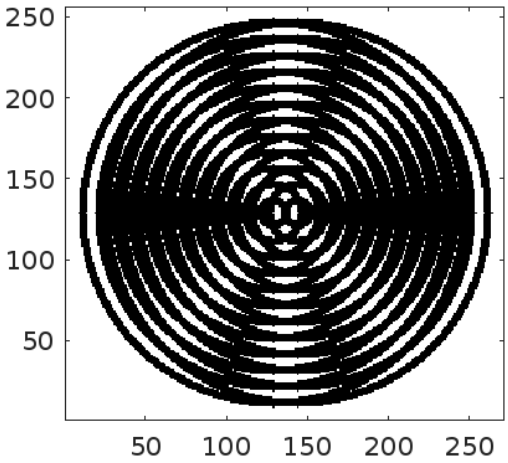

Figure 45: Desplazamiento 15 pixeles

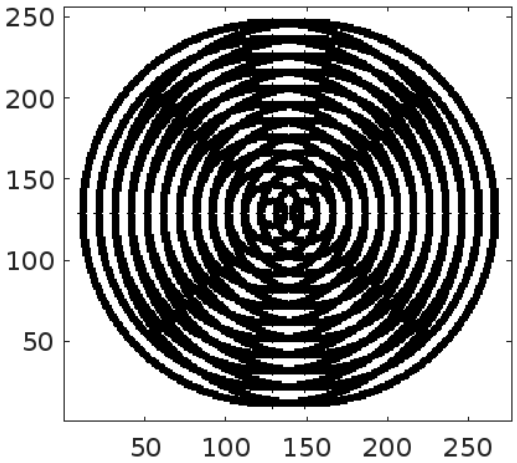

Figura 46: Desplazamiento 20 pixeles

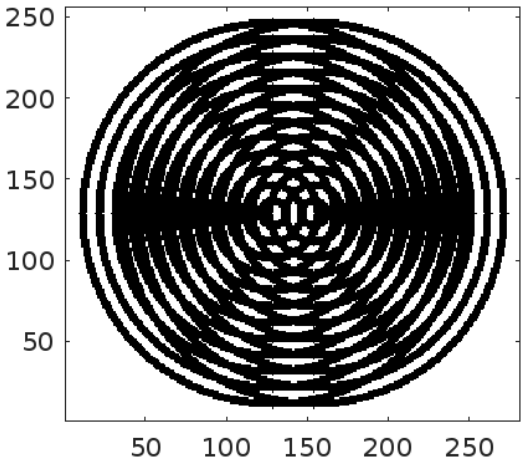

Figure 47: Desplazamiento 25 pixeles

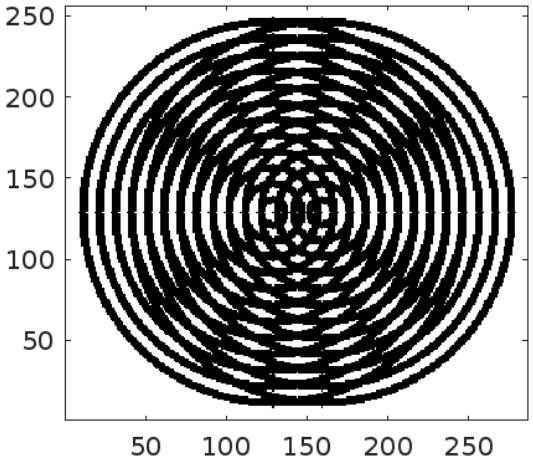

Figura 48: Desplazamiento 30 pixeles

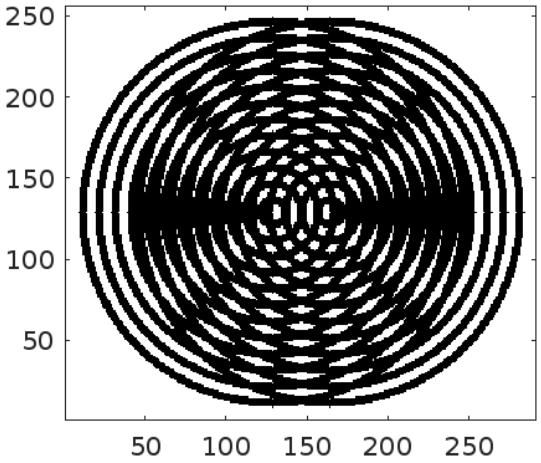

Figure 49: Desplazamiento 35 pixeles

5.1 Código *MatLab* para Anillos concéntricos

```
%-------- Patron superposicion dos conjuntos de  anillos concentricos------

% Creamos un circulo grande, luego le restamos uno mas pequeño y luego
% sumamos uno mas pequeño que el anterior, esto da la ilusion de hacer
% anillos.
px=256; % Tamaño de la imagen en pixeles
diam=240; % Diametro del circulo
A=double(Xcirc2d(px,diam)); % Imagen 256x256 px con circulo de diam 240

% Hacemos los anillos
noanillos=12;       % Especificar el numero de anillos
grosor=10;          % Especificar el grosor de cada anillo

% Este for suma y resta circulos para dejar solo anillos
until=(noanillos*2)-1;
for i=1:until
   A=A+((-1)^i)*double(Xcirc2d(px,diam-(grosor*i))) ;
end

% Hasta aquí el fondo de la imagen es negra y los anillos blancos, para una
% mejor visualizacion invertimos los colores.
A=double(~A);

% Definimos la segunda rejilla igual a la primera
B=A;

% Podemos previsualizar la rejilla obtenida
figure
showImage(A)

% Para crear la ilusion de un desplazamiento agregamos unos a la derecha
% del patron A y unos a la izquierda de B.
despx=20;       % Escoja un desplazamiento en pixeles
O = ones(px,despx);
A=[A,O];
B=[O,B];
[Ay,Ax]=size(A);
[By,Bx]=size(B);

% Obtenemos las transformadas de las rejillas
T_A=fft2(A)/(Ax*Ay);    % Transformada rejilla 1
T_B=fft2(B)/(Bx*By);    % Transformada rejilla 2
T_sA=fftshift(T_A);        % Desplazamiento de Transformada 1 al centro
T_sB=fftshift(T_B);        % Desplazamiento de Transformada 2 al centro

% Convolucionamos las transformadas de las rejillas.
% La funcion de Matlab conv2 no sirve para otras superposiciones
% Debido a la forma de parchar las matrices, nos conviene usar
% la funcion convolve2 que se encuentra en el repositorio de Matlab
Conv = convolve2(T_sA,T_sB,'circular'); % Convolucion de Transformadas
```

```
adc=max(max(abs(Conv))); % Nos servirá de referencia para saber el tamaño
                         % amplitudes con las que trabajamos.

% Para mostrar el resultado de superponer las rejillas, aplicamos la
% Transformada inversa de Fourier y graficamos.
TI_AB=abs(ifft2(Conv));       % Transformada inversa de Fourier de la Convolucion
figure
showImage(TI_AB)

%{
%-----OPCIONAL 1: GRAFICAS ESPECTROS
T_aA=abs(T_sA);              % Amplitud de la Transformada 1
T_aB=abs(T_sB);              % Amplitud de la Transformada 2
Conv_a = abs(Conv);          % Amplitud de la Convolucion
figure
showImage(log(T_aA))
figure
showImage(log(T_aB))
figure
showImage(log(Conv_a))
%}

%{
%-------OPCIONAL 2: FILTRADO-------------
% Podemos filtrar frecuencias de la convolución para solo quedarnos con las
% frecuencias que más influyen en los patrones.
% La función filtro requiere de entradas: una amplitud crítica (a_cr), un
% radio para eliminar TODAS las frecuencias fuera del círculo de dicho
% radio (rad), la matriz de la transformada de la primera rejilla (mT1),
% la matriz de la transformada de la segunda rejilla (mT2) y devuelve la
% matriz del espectro filtrado.
Conv_filtrada=filtrocir(adc*0.1,30,T_sA,T_sB);
% Pedimos la imagen del espectro con el filtro aplicado, dado que las
% amplitudes son pequeñas, conviene visualizar el espectro en escala
% logaritmica
figure
showImage(log(abs(Conv_filtrada)))
% Pedimos la imagen de la Transformada inversa de Fourier del espectro filtrado.
figure
showImage(abs(ifft2(Conv_filtrada)))
%}

%{
%-----------------OPCIONAL 3: GIF PUNTOS +ANGULO-----------------------
% Este codigo hace un gif de las rejillas superpuestas y cambia el angulo
% de la segunda rejilla. Definimos la cantidad de pasos que tendrá la
% animación en tstep.
tstep=5;
for i= 0:tstep
    despx=20*tstep-20*i;       % Escoja un desplazamiento en pixeles
    O = ones(256,despx);
    Agif=[A,O];
    Bgif=[O,B];
    showImage(Agif.*Bgif)
```

```
   frame =getframe(gcf);
   img =  frame2im(frame);
   [img,cmap] = rgb2ind(img,256);

   if i == 0
       imwrite(img,cmap,'Moire_ring_x+.gif','gif','DelayTime',0.1,'LoopCount',Inf);
   else
       imwrite(img,cmap,'Moire_ring_x+.gif','gif','WriteMode','append');
   end
end
%}


function [c2d] = Xcirc2d(pts,diam,k)
% funcion para definir un arreglo con la digitalizacion de la funcion
% circ() en 2 dimensiones f(r) = circ(r/wid)
% r*r = x*x + y*y
% pts*pts es el numero de puntos totales en la matriz,
% y pts debe ser una potencia de 2: 2,4,8,16,32,64,128,.....
% diam es el numero de puntos en el diametro del circulo
x = -pts/2:pts/2-1;
y = -pts/2:pts/2-1;
[xg2d,yg2d] = meshgrid(x,y);   % crear matriz de valores x,y para la funcion gauss()
if ~exist('k','var')
   k=0;
end
if k>=0
   c2d = (xg2d.*xg2d+yg2d.*yg2d) <= (diam*diam/4.0);   % crear matriz de la funcin
circ() en 2d
end
if k<0
   c2d = (xg2d.*xg2d+yg2d.*yg2d) > (diam*diam/4.0);
end
end


function [Conv_F] = filtrocir(a_cr,rad,mT1,mT2)
% Función para hacer filtrado, sólo deja las amplitudes por debajo de una
% amplitud crítica dentro del circulo de radio rad, se filtran desde las
% transformadas originales necesita la amplitud crítica (a_cr), el radio
% en pixeles del círculo de prefiltrado (rad) y las transformadas de las
% rejillas originales (mt1, mT2)
[ypx1,xpx1]=size(mT1);
[ypx2,xpx2]=size(mT2);
mConv = convolve2(mT1,mT2,'circular');

% Más adelante necesitaremos reincorporar el valor maximo de la convolución
% de espectros, por lo que lo guardamos.
mConv_a=abs(mConv);
adc=max(max(mConv_a));

% Para comparar con la amplitud crítica, obtenemos las amplitudes de las
% transformadas
```

```
amT1=abs(mT1); % Amplitud transformada 1
amT2=abs(mT2); % Amplitud transformada 2

% Definimos matrices de unos para hacer los filtros
T_filt1=ones(ypx1,xpx1);
T_filt2=ones(ypx2,xpx2);

% Prefiltramos la imagen despreciando TODAS las frecuencias fuera del
% circulo de radio rad.
for i=1:ypx1
    for j=1:xpx1
        if (j-ceil(xpx1/2)).^2+(i-ceil(ypx1/2)).^2>rad^2
            T_filt1(i,j)=0;
        end
    end
end

for i=1:ypx2
    for j=1:xpx2
        if (j-ceil(xpx2/2)).^2+(i-ceil(ypx2/2)).^2>rad^2
            T_filt2(i,j)=0;
        end
    end
end
for i=1:ypx1             % Eliminamos las frecuencias de las rejillas base
   for j=1:xpx1
       if amT1(i,j)>a_cr
           T_filt1(i,j)=0;
       end
   end
end
for i=1:ypx2
    for j=1:xpx2
       if amT2(i,j)>a_cr
           T_filt2(i,j)=0;
       end
   end
end

Conv_F = mConv.*T_filt1.*T_filt2; % Filtramos la convolucion de imagenes
[ypxc,xpxc]=size(Conv_F);
Conv_F(ceil((ypxc/2)+1),ceil((xpxc/2)+1))=adc; % Reincorporamos la amplitud maxima
end

function showImage(img,x,y)
   if ~exist('x','var')
       x = 1:size(img,2);
   end
   if ndims(x)==2
       x = x(1,:);
   end
   if ~exist('y','var')
       y = 1:size(img,1);
   end
```

```
    if ndims(y)==2
        y = y(:,1)';
    end

    imagesc(x,y,img)
     axis equal
     axis tight
    colormap(gray);
    set(gca,'YDir','normal')
end
```

6. Números del 0 al 9

Además de los patrones presentados con anterioridad, se puede realizar animaciones con múltiples imágenes separadas por una rejilla cuadrada. Esto queda mejor ilustrado con los ejemplos que se presentan a continuación, empezando con una imagen con los números de 0 a 9.

Supóngase que se desea obtener la imagen del número cero (Figura 50) como resultado de la superposición de dos rejillas. Nótese que todos los números utilizados en esta sección fueron hechos en un programa de dibujo (nosotros utilizamos el software libre *InkScape* pero se puede utilizar cualquier programa de dibujo). Este método permite el control del tipo de font y el tamaño de los números, para asegurar que todas las imágenes utilizadas son compatibles entre sí para la aplicación de Moiré. Cada número debe ser guardado en un archivo de imagen por separado, y los archivos de estas imágenes debe ser accesibles al programa en MatLab.

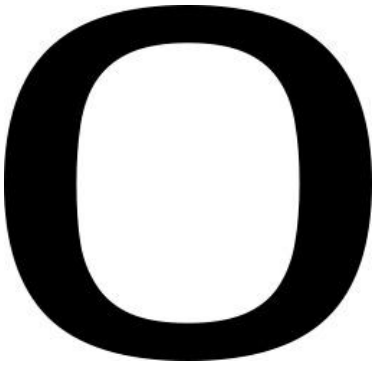

Figura 50: Número 0.

Si se establece que una de las dos rejillas que formará dicha imagen es una rejilla cuadrada, como se muestra en la Figura 51. Esta rejilla tiene renglones negros de 9 pixeles de ancho y renglones blancos de 1 píxel de ancho. (Estos valores vienen del hecho que hay 10 números de 0 a 9, y en cada posición de la rejilla cuadrada se requiere bloquear 9 de estos números y dejar pasar a 1.)

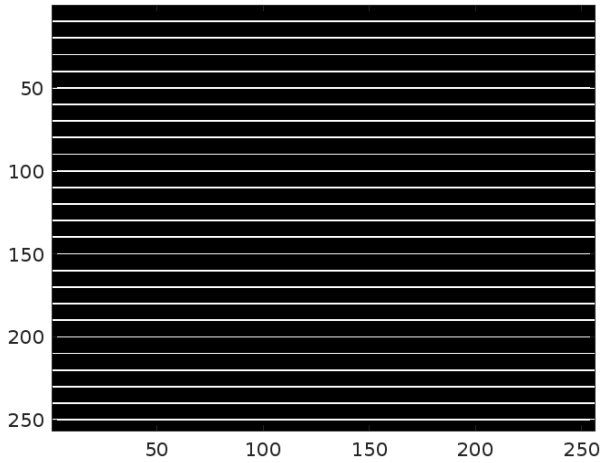

Figura 51: Rejilla cuadrada con renglones negros de 9 pixeles de ancho y renglones blancos de 1 píxel de ancho.

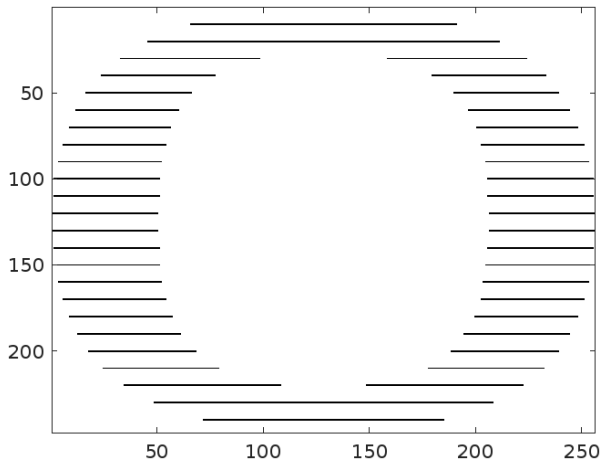

Figura 52: Rejilla número 0 franjeado.

Entonces la rejilla que forma la silueta del cero es la de la Figura 52. Esta rejilla está hecha de líneas de 1 píxel de ancho separado por 9 pixeles, para que empate con la rejilla cuadrada de la Figura 51. Lo que es equivalente, se puede pensar que en la Figura 50 se quitan renglones de 9 pixeles de ancho, y se deja la información en renglones de 1 píxel de ancho.

Ahora, al superponer Fig. 52 con Fig. 51 se obtiene la imagen mostrada en la Figura 53.

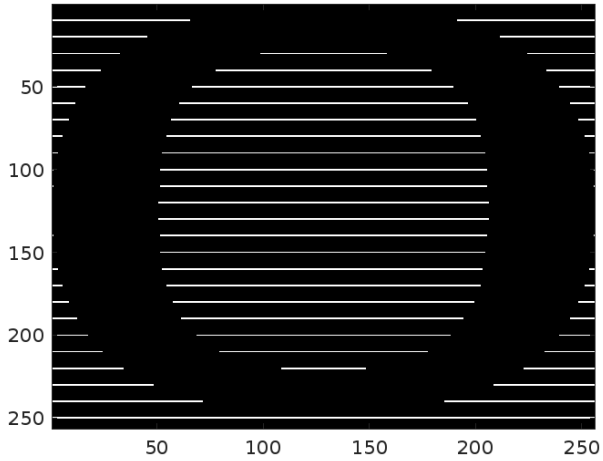

Figura 53: Número 0 formado por la superposición de las rejillas de las Figuras 51 y 52.

Lo mismo se puede hacer para cualquier número del 0 al 9. Las Figuras 54 a 62 muestran las rejillas de todos los números franjeados. Nótese que todas estas rejillas usan la misma forma de producir la rejilla: cada rejilla está hecha de líneas de 1 píxel de ancho separado por 9 pixeles, para que empate con la rejilla cuadrada de la Figura 51.

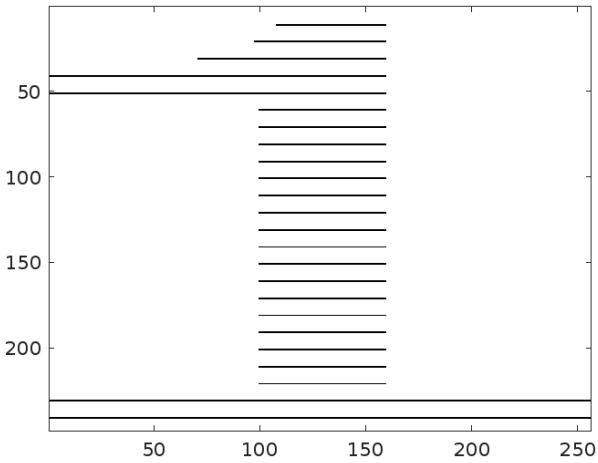

Figura 54: Rejilla número 1 franjeado

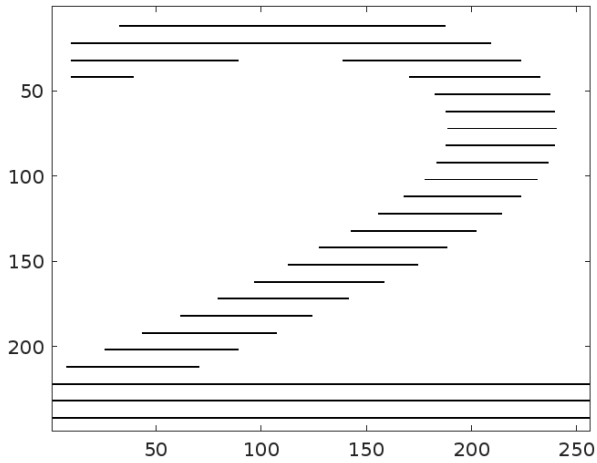

Figura 55: Rejilla número 2 franjeado

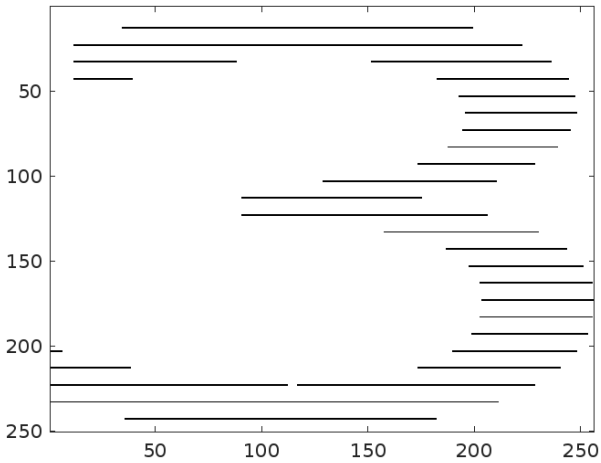

Figura 56: Rejilla número 3 franjeado

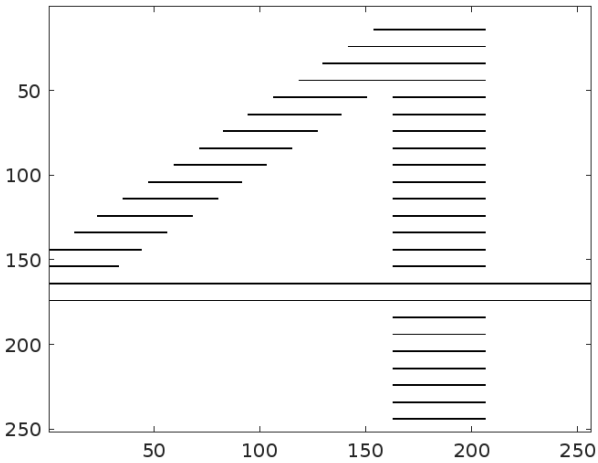

Figura 57: Rejilla número 4 franjeado

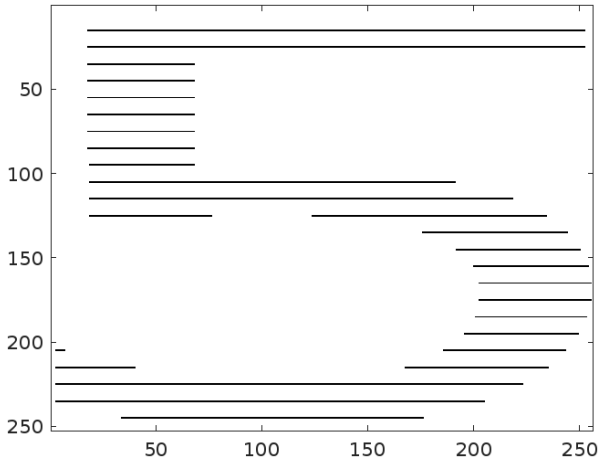

Figura 58: Rejilla número 5 franjeado

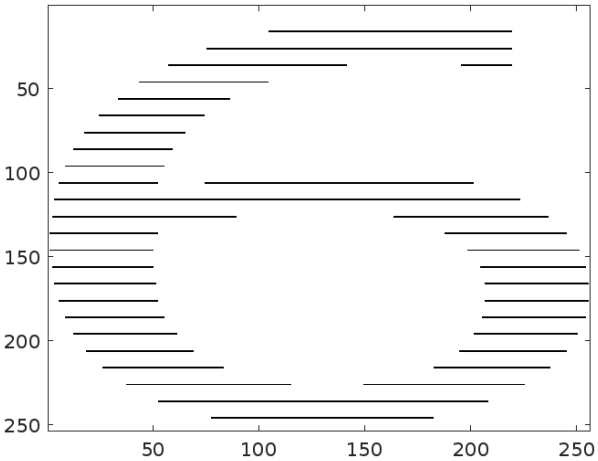

Figura 59: Rejilla número 6 franjeado

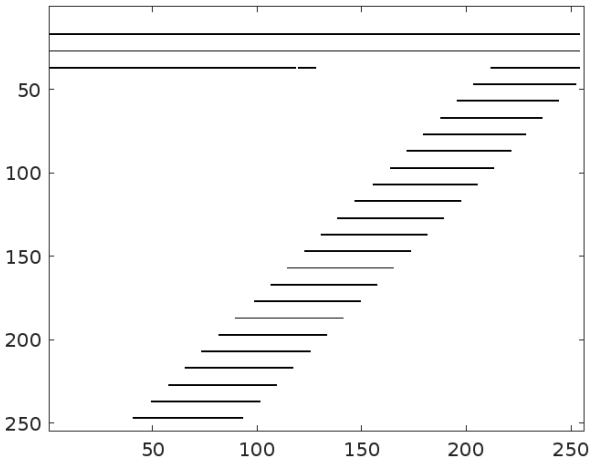

Figura 60: Rejilla número 7 franjeado

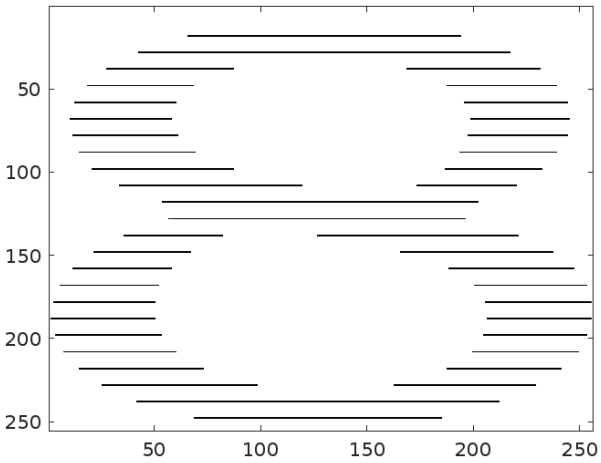

Figura 61: Rejilla número 8 franjeado

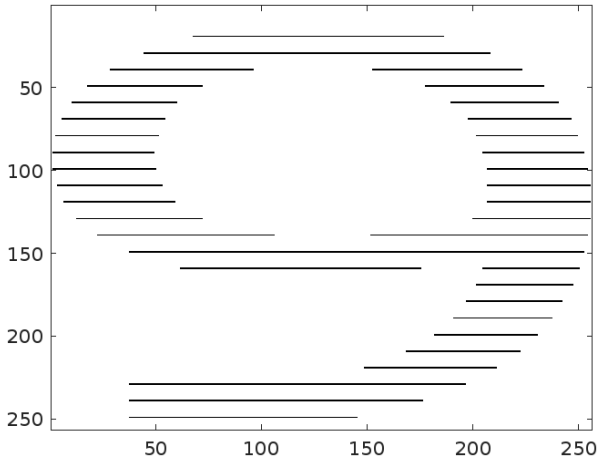

Figura 62: Rejilla número 9 franjeado

Ahora, está claro que cualquiera de las rejillas de números formará la imagen del número al superponerla con la rejilla cuadrada de Figura 51, entonces la pregunta es ¿hay manera de comprimir toda esta información en una sola rejilla? La respuesta es sí. Tómese la rejilla de 0 de la Figura 52 y combínela con la rejilla del número 1, Figura 54, de tal forma que las franjas del 1 queden un píxel por debajo de las franjas del 0, luego agregue la rejilla del número 2, la Figura 55, igualmente un píxel abajo de las franjas del número anterior, el 1, y así consecutivamente hasta combinar las diez rejillas. La rejilla final con todos los números se verá como se muestra en la Figura 63.

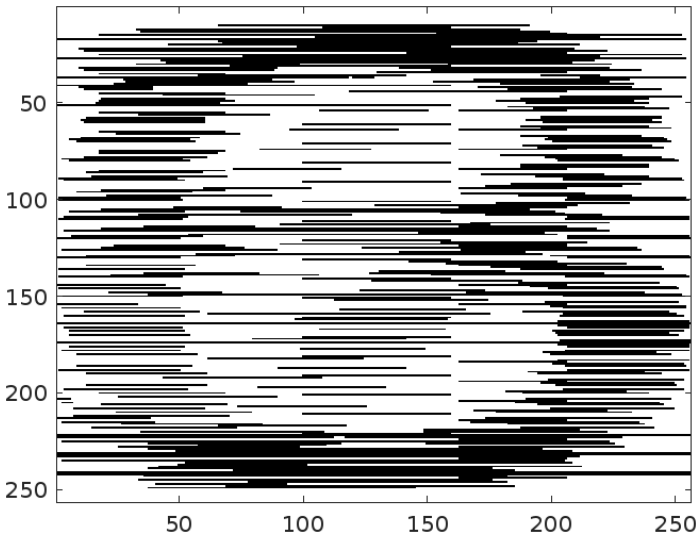

Figura 63: Rejilla de todos los números del 0 al 9.

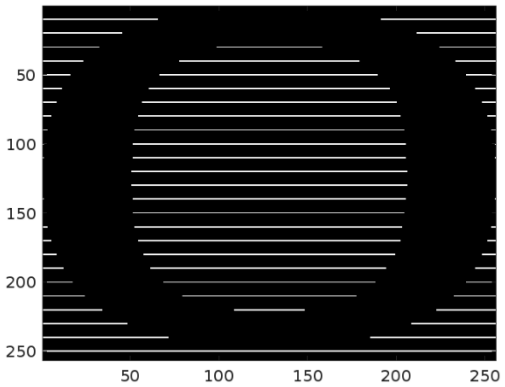

Figura 64: Superposición con desplazamiento 0 pixeles

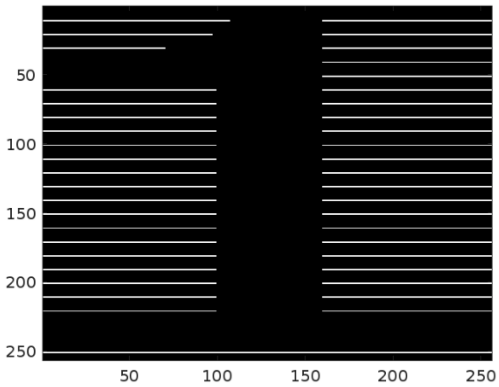

Figura 65: Superposición con desplazamiento 1 píxel

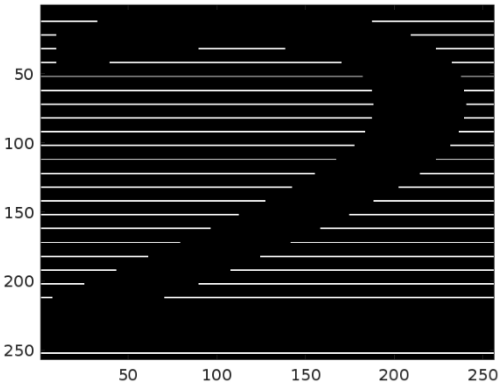

Figura 66: Superposición con desplazamiento 2 pixeles

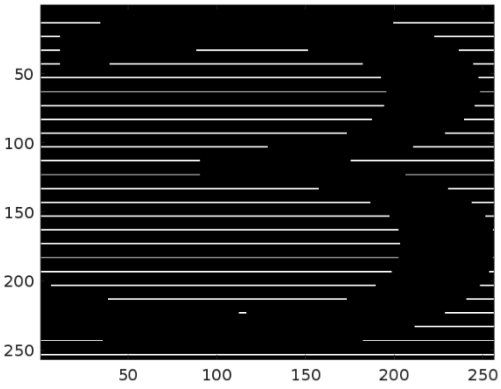

Figura 67: Superposición con desplazamiento 3 pixeles

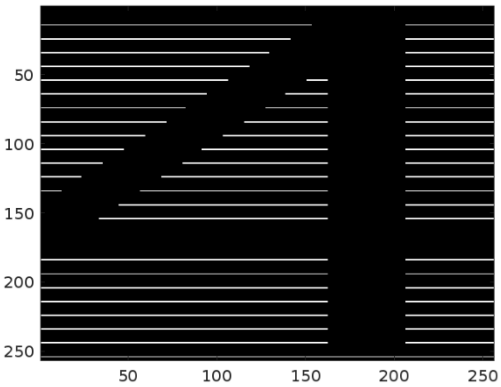

Figura 68: Superposición con desplazamiento 4 pixeles

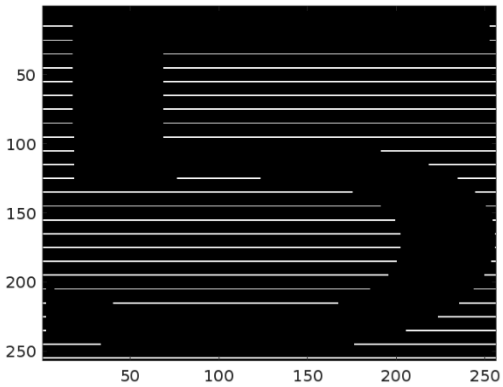

Figura 69: Superposición con desplazamiento 5 pixeles

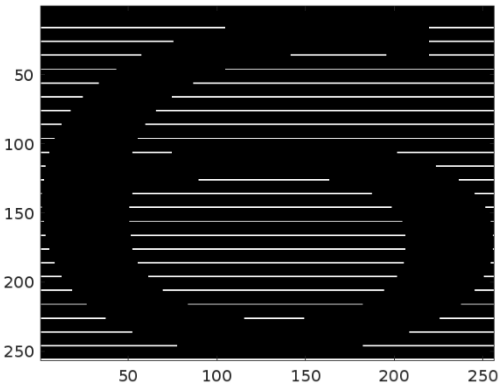

Figura 70: Superposición con desplazamiento 6 pixeles

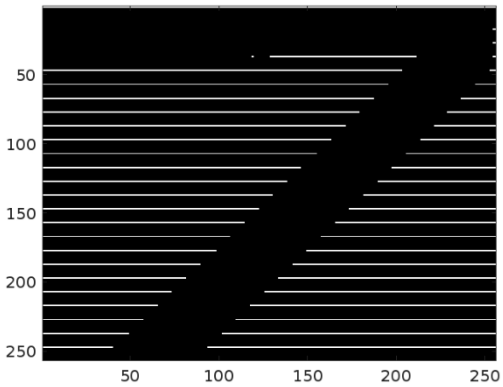

Figura 71: Superposición con desplazamiento 7 pixeles

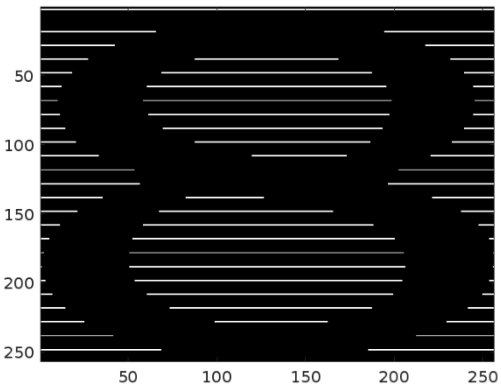

Figura 72: Superposición con desplazamiento 8 pixeles

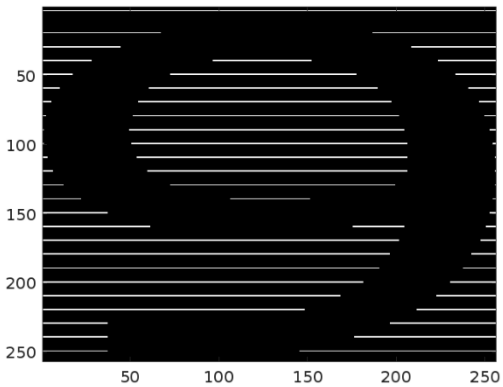

Figura 73: Superposición con desplazamiento 9 pixeles

La rejilla de la Figura 63 contiene toda la información de los números, entonces la cuestión ahora es ¿cómo hacer que regrese esa información? Si se le coloca de nuevo la rejilla cuadrada de Figura 51, alineada desde la parte superior, la imagen que se forma está dado por la Figura 64, que muestra el número 0. Pero al mover la rejilla cuadrada de la Figura 51 un píxel más abajo, la imagen que aparece es la de la Figura 65, que es el número 1. De esta forma los desplazamientos en pasos de 1 píxel van formando las imágenes de las Figuras 66 al 73. El desplazamiento indicado en las leyendas de las figuras es el desplazamiento desde la primera posición, para ver el número 0. Estas figuras muestran claramente la razón de haber elegido que la rejilla cuadrada tuviera renglones negros de 9 pixeles de ancho y renglones blancos de 1 píxel de ancho, pues esto permite que se oculte la información de todos los números excepto uno, dicha información pasa a través del renglón blanco y forma la imagen deseada.

Hasta ahora tenemos una rejilla sensible a movimientos, pero estos movimientos están restringidos a números enteros de pixeles debido a que estos son los movimientos más pequeños que se puede hacer entre una imagen y el otro en la computadora. Sin embargo, si se buscara relacionar la distancia que se desplaza la rejilla cuadrada con la imagen que se está formando, resultaría conveniente tener acceso a movimientos más finos para hacer un perfil de intensidades, particularmente para la aplicación de un vernier de Moiré. Dadas las condiciones, ¿hay alguna forma de obtener movimientos más finos en la simulación? La respuesta es sí, utilizando la transformada de Fourier.

Considérese una función $f(x,y)$ con transformada de Fourier $F(u,v)$, por el teorema de corrimiento en la teoría de la transformada de Fourier, se tiene que un desplazamiento espacial en la función original influye en la transformada de la siguiente manera (Goodman, 1996):

$$F\big(f(x-x_0, y-y_0)\big) = F(u,v)e^{i2\pi(x_0u+y_0v)}. \tag{4}$$

De esta manera se puede mover una rejilla desde su espectro, agregando un término de fase a la transformada de Fourier de una de las dos rejillas, y la ventaja que ofrece desde el punto de vista computacional es que el movimiento ya no queda restringido a pixeles enteros, porque los parámetros x_0 y y_0 pueden definirse como valores fraccionales.

De esta manera se puede obtener desplazamientos de medio píxel (ver las Figuras 74 a 79), o de cuarto de pixel (ver las Figuras 80 a 83).

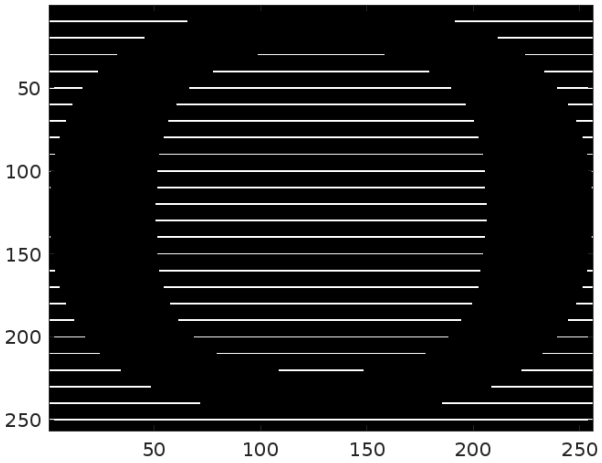

Figura 74: Desplazamiento 0 pixeles

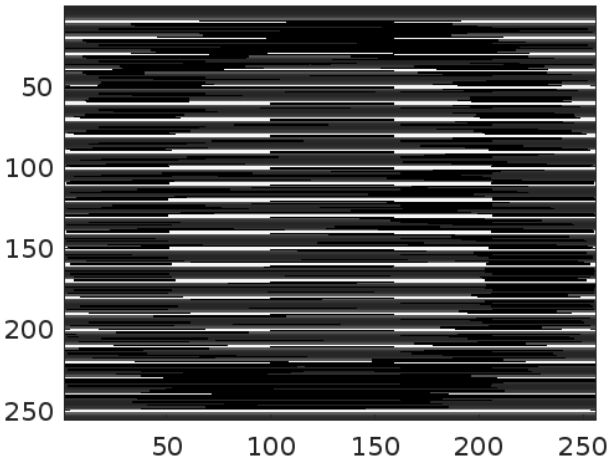

Figura 75: Desplazamiento $1/2$ píxel

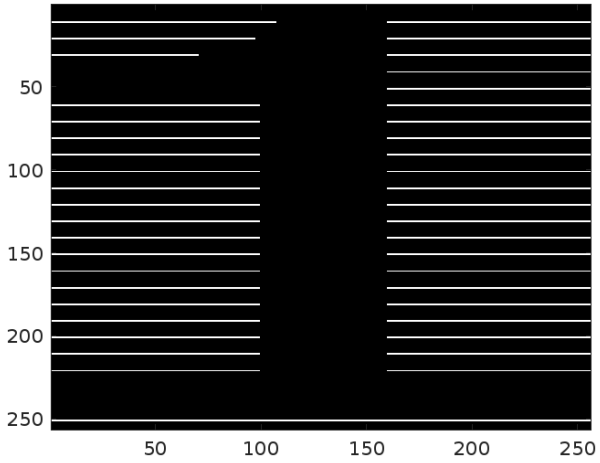

Figura 76: Desplazamiento 1 píxel

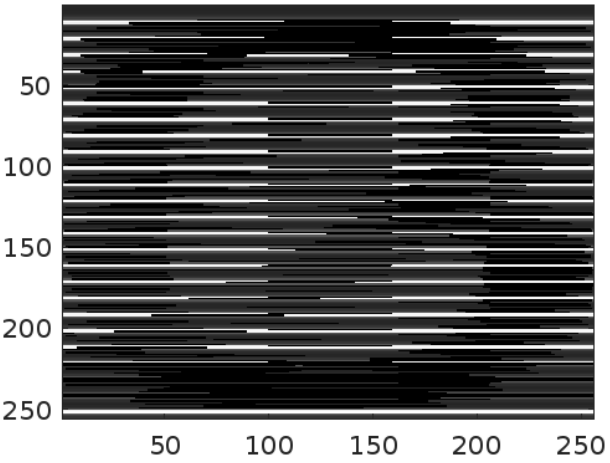

Figura 77: Desplazamiento $3/2$ pixeles

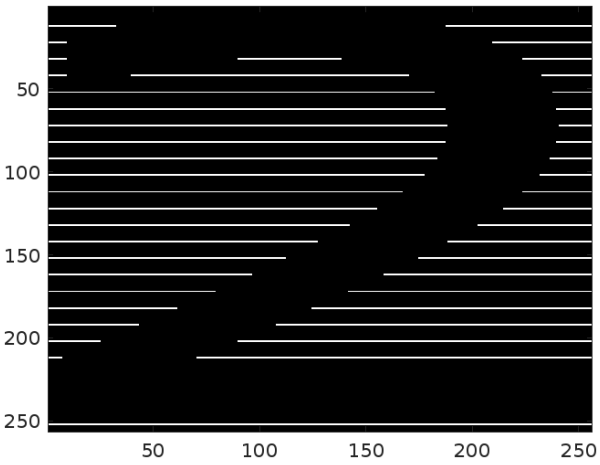

Figura 78: Desplazamiento 2 pixeles

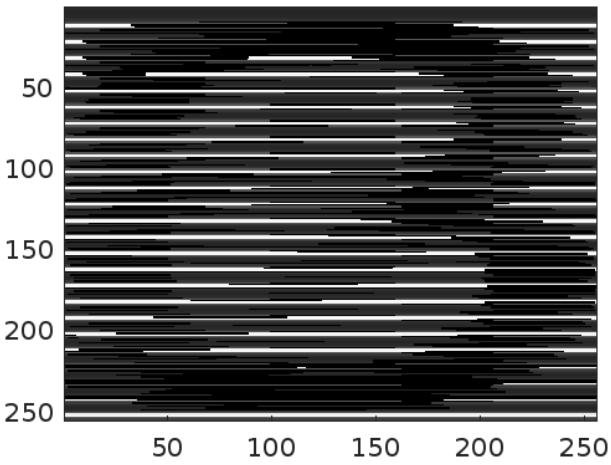

Figura 79: Desplazamiento $5/2$ pixeles

Nótese que las figuras que corresponden a movimientos menores de un píxel presentan "sombras" de dos números consecutivos; en la Figura 75, el desplazamiento se encuentra a la mitad del camino entre el 0 y el 1, por lo que se puede apreciar la sombra de ambos números en esta posición. Lo mismo ocurre en las Figuras 77 y 79, con los números 1 y 2, y, 2 y 3, respectivamente.

Para los desplazamientos de $1/4$ de píxel, también se aprecian las sombras de los números 0 y 1, sólo que a $1/4$ de píxel, la sombra predominante es la del 0 con una sombra débil del 1 y, contrariamente, en el paso de $3/4$ de pixel la sombra que predomina es la del 1, con una sombra débil de 0.

Con estas ideas en mente, este tipo de rejillas pueden ser útiles para medir distancias en el mundo real, pues cada posición desplegaría un perfil de intensidad distinto, y si éste está bien caracterizado, podría hacerse una relación posición-intensidad precisa. Esta es la base del vernier de Moiré propuesto en (Reid, 1984).

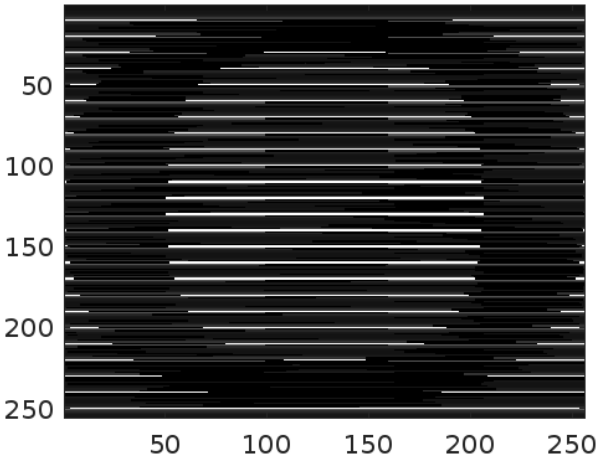

Figura 80: Desplazamiento $1/4$ píxel

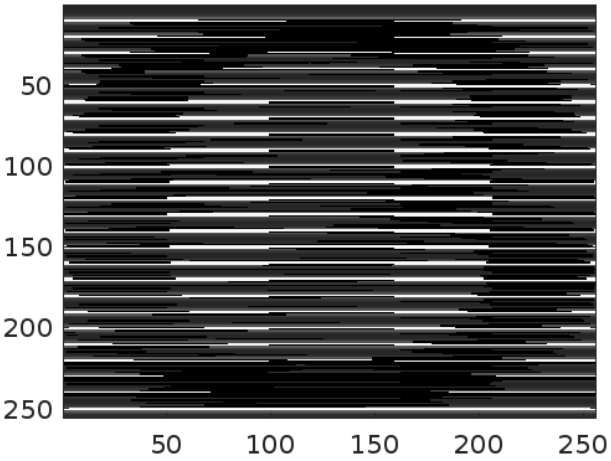

Figura 81: Desplazamiento $1/2$ píxel

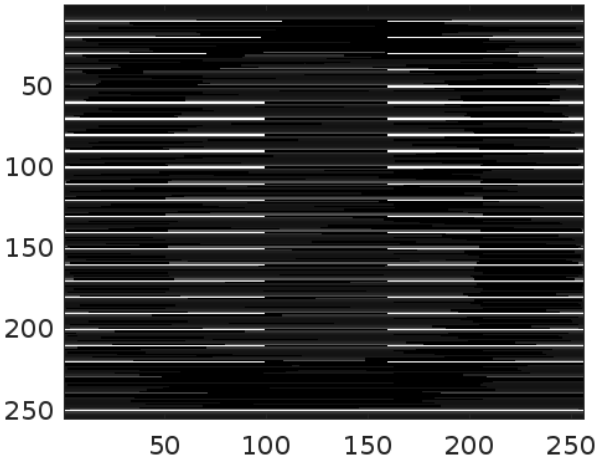

Figura 82: Desplazamiento $3/4$ píxel

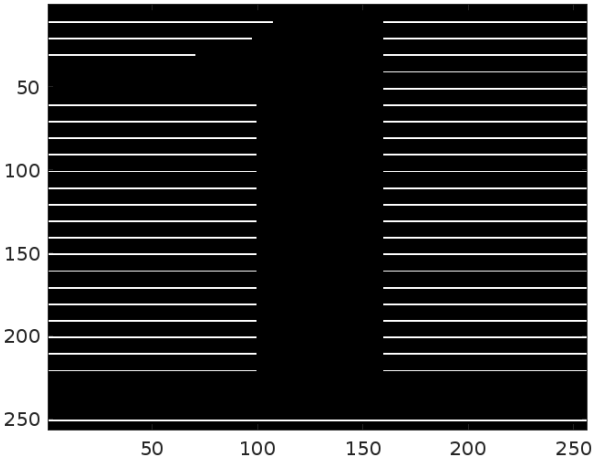

Figura 83: Desplazamiento 1 píxel

6.1 Código *MatLab* para números del 0 al 9

```
%------------------------ANIMACION NUMEROS------------------
% Para hacer animacion con Moire buscamos formar figuras al mover una
% rejilla de onda cuadrada por lo que cada frame debe acompletarse con
% dicha rejilla. Le quitaremos franjas a las imagenes originales con un
% desfase de un pixel y luego combinaremos las imagenes para
% hacer una sola rejilla y al ponerle una rejilla cuadrada se formara la
% imagen del numero y al moverla hacia abajo aparecerá el siguiente numero.


% Leemos las imagenes a utilizar para la animacion y las ponemos en formato
% blanco y negro.
```

```
% Para este programa se supone que las imágenes de cada número están en un
% archivo de imagen por separada, y que están guardadas en un path que es
% accesible para este programa. Debe asegurarse que el nombre del archivo
% y el path dentro del comando “imread()” es correcto para cada caso.

A{1}=double(im2bw(imread('Informe/NUMEROS/0.jpg'),0.4));
A{2}=double(im2bw(imread('Informe/NUMEROS/1.jpg'),0.4));
A{3}=double(im2bw(imread('Informe/NUMEROS/2.jpg'),0.4));
A{4}=double(im2bw(imread('Informe/NUMEROS/3.jpg'),0.4));
A{5}=double(im2bw(imread('Informe/NUMEROS/4.jpg'),0.4));
A{6}=double(im2bw(imread('Informe/NUMEROS/5.jpg'),0.4));
A{7}=double(im2bw(imread('Informe/NUMEROS/6.jpg'),0.4));
A{8}=double(im2bw(imread('Informe/NUMEROS/7.jpg'),0.4));
A{9}=double(im2bw(imread('Informe/NUMEROS/8.jpg'),0.4));
A{10}=double(im2bw(imread('Informe/NUMEROS/9.jpg'),0.4));

Asize=size(A);
noIm=Asize(1,2); % Este parametro guarda el numero de imagenes que hay

% Ajustamos el tamaño de las imagenes para que al combinarlas, la imagen
% resultante sea de 256x256 (tamaño estandar)

for i=1:noIm
   A{i}=imresize(A{i},[256-(noIm-1),256]);
end

% Con imgrid generamos las franjas en las imagenes, y con step agregamos
% una fila blanca al inicio de cada numero para que al combinar las
% imagenes las lineas de cada numero no se estorben entre si.
for i=1:noIm
   AG{i}=imgrid(A{i},noIm-1,1);
   AG{i}=step(AG{i},(i-1));
end
% Pedimos ver un ejemplo
figure
showImage(AG{1})

% La funcion alignmat va a alinear las matrices que representan las
% imagenes y va a formar una matriz que tenga la informacion de las
% anteriores. El for empieza en 2 porque la primera imagen es la base.
AN=AG{1};
for i=2:noIm
   AN=alignmat(AG{i},AN);
end
% Pedimos ver la rejilla final con toda la informacion de las anteriores
figure
showImage(AN)

% Guardamos las medidas de la rejilla final AN para utilizarlo mas abajo.
sAN=size(AN);

% Generamos la malla que acompletara cada uno de los numeros
rc=vargrid(noIm-1,1,sAN(1,1),sAN(1,2));
src=size(rc); % Guardamos sus dimensiones
```

```
% Pedimos una visualización de la rejilla
figure
showImage(rc)

% Ahora hagamos el desplazamiento de la rejilla anterior desde el espacio
% de frecuencias para esto consideremos que un desplazamiento se ve como un
% cambio de fase: A*e^(i*2*pi*u/N)

% Declaramos la Transformada de las rejilla cuadrada original y la rejilla
% de numeros,Trc y TAN respectivamente.
T_rc{1}=fftshift(fft2(rc)/(src(1,1)*src(1,2)));
T_AN=fftshift(fft2(AN)/(sAN(1,1)*sAN(1,2)));

% IMPORTANTE: Defina el movimiento en pixeles que desea, nótese que puede
% pedir movimientos de menos de un pixel, p.e. 1/4 de pixel
dpx=1;

% Notese que si dpx es diferente de 1, entonces aumenta o dismnuye el
% número de pasos que necesitamos para visualizar toda la rejilla, si es
% menor a 1 necesitaremos más pasos, si es mayor a 1 entonces necesitaremos
% menos pasos.
until=noIm*(1./dpx);

% El for empieza en 2 porque no hace falta mover la rejilla para el primer
% numero y aqui se desplazan las rejillas.
for i=2:1:until
   for j=1:sAN(1,2)
       for k=1:sAN(1,1)
           T_rc{i}(j,k)=T_rc{1}(j,k)*exp(-1i*2.*pi*j*(dpx)*(i-1)/sAN(1,2));
       end
   end
end

% En este for se convolucionan la rejilla de numeros con las rejillas de
% franjas desplazadas.
for i=1:1:until
   Conv_TAN=convolve2(T_AN,T_rc{i},'circular');
   TI=abs(ifft2(Conv_TAN));
   fig=figure;
   showImage(TI)

   %
   %-------OPCIONAL: GIF ----------------------------------------
   % Ponga una { en el % libre de arriba si no desea que se guarde el GIF
   % Ahora hacemos un gif con las imágenes obtenidas de los números
   frameo = getframe(fig);
   im = frame2im(frameo);
   [imind,cm] = rgb2ind(im,256);
   outfile = 'numeros19.gif';
   % On the first loop, create the file. In subsequent loops, append.
   if i==1
       imwrite(imind,cm,outfile,'gif','DelayTime',1,'loopcount',inf);
   else
       imwrite(imind,cm,outfile,'gif','DelayTime',1,'writemode','append');
```

```
   end
   %}
end

function [imgd] = imgrid(mat,b,w)
% Funcion que borra renglones de una imagen mat, los renglones son de ancho
% w pixeles y estan espaciados por b pixeles.
   s=size(mat);
   imgd=mat;
   j=1;
   bw=b+w;
% Checamos que los parámetros b y w no excedan las dimensiones de mat
   if b>s(1,1) || w>s(1,1)
       fprintf('check size')
% En caso de que los parametros sean viables, procedemos con la funcion
   else
       for i=0:s(1,1)      % Nos ubicamos en la 1ra entrada de la matriz
           j=i*bw+1;       % Esto nos ubica en donde iniciara el renglon
           if j>s(1,1)     % Verificamos que seguimos dentro de la matriz
               break       % En caso contrario acabamos el proceso
          end
          while j<=b+(i*bw) % Nos mantiene dentro del renglon que borraremos
               if j>s(1,1)
                   break   % Verificamos que seguimos dentro de la matriz
               end
               imgd(j,:)=1;     % Aquí borramos todo el renglon actual
               j=j+1;           % Pasamos al siguiente renglon de la matriz
           end
       end
   end
end

function [grid]=vargrid(b,w,xpix,ypix)
% Esta funcion hace rejillas cuadradas con renglones negros de b pixeles de
% ancho y renglones blancos de w pixeles de ancho. La rejilla tiene xpix
% columnas y ypix renglones.
   grid=ones(ypix,xpix);  % Definimos una matriz inicial de 1's
% Y con el mismo codigo de la funcion imgrid, le hacemos los renglones
% negros
   j=1;
   bw=b+w;
   if b>ypix || w>ypix
       fprintf('check size')
   else
       for i=0:ypix
           j=i*bw+1;
           if j>ypix
               break
           end
           while j<=b+(i*bw)
               if j>ypix
                   break
               end
               grid(j,:)=0;
```

```
                j=j+1;
            end
        end
    end
end

function [mats]=step(mat,pix)
% Esta funcion agrega renglones blancos al inicio de una matriz mat, el
% renglon tiene ancho pix pixeles.
    s=size(mat);
    mat0=ones(pix,s(1,2)); % Hacemos una matriz de 1's con el ancho deseado
    mats=vertcat(mat0,mat); % La colocamos arriba de la matriz mat
end

function [amat]=alignmat(mat1,mat2)
% Esta funcion combina dos matrices mat1 y mat2 desde sus centros esta
% funcion esta pensada para combinar matrices que representen imagenes a
% b&w cuya informacion no se superpone, esto es que en una misma celda solo
% una de las matrices puede ser negra (0).
% La idea para combinar las matrices es sumar los valores de una misma
% celda en cada matriz para obtener el de la matriz combinada, pero como
% blanco mas blanco debe ser blanco, y esto es 1 + 1 = 1, entonces mejor
% intercambiamos los 1 por 0 y viceversa, de esta forma los pixeles blancos
% ahora son 0 y 0+0 = 0 y en caso de que haya negro en una imagen y en la
% otra blanco, tendriamos 1+0=1 (negro)
    mat1=double(~mat1);
    mat2=double(~mat2);
    s1=size(mat1);
    s2=size(mat2);
    dsy=s1(1,1)-s2(1,1);  % Las matrices pueden tener distinto tamaño por lo
    dsx=s1(1,2)-s2(1,2);  % que guardamos el valor de esas diferencias.
    if dsy==0 & dsx==0    % Si tienen la misma dimension, la combinacion es
        amat=mat1+mat2;   % la suma directa de ambas matrices
    else
        m=1;
        n=1;
        if dsy<0          % Si son de distinto tamaño, localizamos la matriz
            m=s2(1,1);    % con mas renglones
        elseif dsy>=0
            m=s1(1,1);
        end
        if dsx<=0         % Localizamos la matriz con más columnas
            n=s2(1,2);
        elseif  dsx>0
            n=s1(1,2);
        end
        % Definimos la matriz que tendrá los valores combinados
        amat=zeros(m,n);
        % La alineación es en el centro pero el calculo del centro de una
        % matriz cambia si las dimensiones son pares o impares, cx1 guarda
        % la coordenada x del centro de mat1, cx2 la coordenada x del centro
        % de mat2. Notese que solo alinearemos con el centro en x porque de
        % acuerdo al objetivo del programa nos interesa que en y no haya
        % desplazamiento alguno, por lo que acomodaremos desde el primer
```

```
        % renglon
         if rem(s1(1,2),2) == 0
            cx1=(s1(1,2)/2)+1;
        elseif rem(s1(1,2),2) == 1
            cx1=ceil(s1(1,2)/2);
        end
        if rem(s2(1,2),2) == 0
            cx2=(s2(1,2)/2)+1;
        elseif rem(s2(1,2),2) == 1
            cx2=ceil(s2(1,2)/2);
        end
        % También localizamos el centro de la matriz combinada amat
        if rem(n,2) == 0
            cxm=(n/2)+1;
        elseif rem(n,2) == 1
            cxm=ceil(n/2);
        end
        % Este for ingresa los valores de mat1 directo en amat pero los
        % coloca de tal forma que el centro de mat1 quede en el centro de
        % amat, de acuerdo al objetivo de esta funcion es importante que en
        % y no se desplacen las matrices por lo que los valores de mat1 se
        % ingresan desde el primer renglon.
        for i=1:s1(1,1)
            for j=1:s1(1,2)
                amat(i,(cxm-cx1)+j)=mat1(i,j);
            end
        end
        % Ahora se mete la información de mat2 a amat, pero no debemos
        % borrar la información ya existente de mat1. Notese que mat2
        % tambien se ingresa de forma centrada.
        for i=1:s2(1,1)
            for j=1:s2(1,2)
                % Si estamos fuera de las celdas que ya tienen info de mat1
                % entonces ponemos directo la info de mat2
                if (cxm-cx2)+j<(cxm-cx1)+1 || (cxm-cx2)+j>(cxm-cx1)+s1(1,2)
                    amat(i,(cxm-cx2)+j)=mat2(i,j);
                else
                    if i>s1(1,1)
                        amat(i,(cxm-cx2)+j)=mat2(i,j);
                % Si estamos en celdas con info de mat1, entonces sumamos el
                % valor preexistente de mat1 con el nuevo de mat2
                    else
                        amat(i,(cxm-cx2)+j)=amat(i,(cxm-cx2)+j)+mat2(i,j);
                    end
                end
            end
        end
    end
    amat=double(~amat); % Regresamos a la escala de colores original
end

function [c1d] = Xcomb1dp(pts,sep,dx)
% funcin para definir un arreglo con la digitalizacin de la funcin
% comb() en 1 dimension. Hay una delta() en el punto central (pts/2)+1.
```

```
% pts es el nmero de puntos totales en el arreglo,
% y debe ser una potencia de 2: 2,4,8,16,32,64,128,.....
% sep es la separacin de cada delta() en la comb()
c1d = zeros(1,pts); % crear arreglo para la funcin delta()
c1d(pts/2+1+dx) = 1;  % el punto central es una delta()
% agregar las otras deltas al arreglo
for i = 1:ceil(pts/(2*sep))+1+dx
   if (pts/2+1+dx+(i*sep)) <= pts
       c1d(pts/2+1+dx+(i*sep)) = 1;
   end
   if (pts/2+1+dx-(i*sep)) >= 1
       c1d(pts/2+1+dx-(i*sep)) = 1;
   end
end
end

function [c2d] = Xcomb2dp(pts,sepx,sepy,dx,dy)
% funcin para definir un arreglo con la digitalizacin de la funcin
% comb() en 2 dimensiones. Hay una delta() en el punto central.
% pts es el nmero de puntos totales en el arreglo,
% y debe ser una potencia de 2: 2,4,8,16,32,64,128,.....
% sepx es la separacin de cada delta() en la comb() en x
% sepy es la separacin de cada delta() en la comb() en y
cx1d = Xcomb1dp(pts,sepx,dx);   % 1d comb() in x
cy1d = Xcomb1dp(pts,sepy,dy);   % 1d comb() in y
% generar una matriz pts*pts con solo una fila = cx1d
cx2d = zeros(pts,pts);
cx2d(pts/2+1+dx,:) = cx1d;
% generar una matriz pts*pts con solo una columna = cy1d
cy2d = zeros(pts,pts);
cy2d(:,pts/2+1+dy) = cy1d.';
% la convolucin en 2d de cx2d con cy2d da la matriz de la comb() en 2d
c2d = conv2(cx2d,cy2d,'full');
end

function [r2d] = Xrect2d(pts,widx,widy)
% funcin para definir un arreglo con la digitalizacin de la funcin
% rect() en 2 dimensiones.
% pts*pts es el nmero de puntos totales en el arreglo,
% y pts debe ser una potencia de 2: 2,4,8,16,32,64,128,.....
% widx es el nmero de puntos en la direccin x con valor 1 y debe
% ser entre 1 y pts
% widy es el nmero de puntos en la direccin y con valor 1 y debe
% ser entre 1 y pts
% si widx o widy no es menor que pts, obligar a que wid = pts-2
if (widx > pts)||(widx < 1)
   widx = pts - 2;
end
if (widy > pts)||(widy < 1)
   widy = pts - 2;
end
x = 1:pts;
y = 1:pts;
[xr2d,yr2d] = meshgrid(x,y);   % crear matriz de valores x,y para la funcin rect()
```

```
minptsx = pts/2 - ceil(widx/2) + 1;    % posicin en la matriz donde empieza valores =
1
minptsy = pts/2 - ceil(widy/2) + 1;
                            % agregar 1s en el centro de la matriz
r2d = ((xr2d >= minptsx) & (xr2d < minptsx + widx) & (yr2d >= minptsy) & (yr2d <
minptsy + widy));
end

function showImage(img,x,y)
   if ~exist('x','var')
       x = 1:size(img,2);
   end
   if ndims(x)==2
       x = x(1,:);
   end
   if ~exist('y','var')
       y = 1:size(img,1);
   end
   if ndims(y)==2
       y = y(:,1)';
   end

   imagesc(x,y,img)
%    axis equal
%    axis tight
   colormap(gray);
end
```

7. Gato saltando

Se pueden rescatar muchas de las ideas del caso anterior para hacer otro tipo de rejillas, en este caso imágenes en movimiento. Como ejemplo de este tipo de animación de Moiré, se ocupó la plantilla de la Figura 84.

Nótese que las imágenes están ordenadas de forma que simulan posiciones consecutivas que tendría un gato que salta hacia adelante. Para realizar la animación, se separa cada imagen individual del gato y se hace una imagen independiente, de esta forma se tienen 9 imágenes que se quiere ver uno después del otro. Análogamente al caso anterior, se considera una rejilla cuadrada para formar las imágenes, pero en esta ocasión, las franjas negras deben ser de 8 pixeles de ancho, esto para ocultar los 8 imágenes franjeadas restantes al paso que se quiere mostrar en cada posición de la rejilla cuadrada, a través de la franja blanca que es de 1 píxel de ancho. También, se sugiere

que, en este caso, la rejilla quede orientada con las franjas verticales (ver la Figura 85), esto bajo la intuición de que el gato avanza hacia la derecha, así que conviene mover la rejilla horizontalmente.

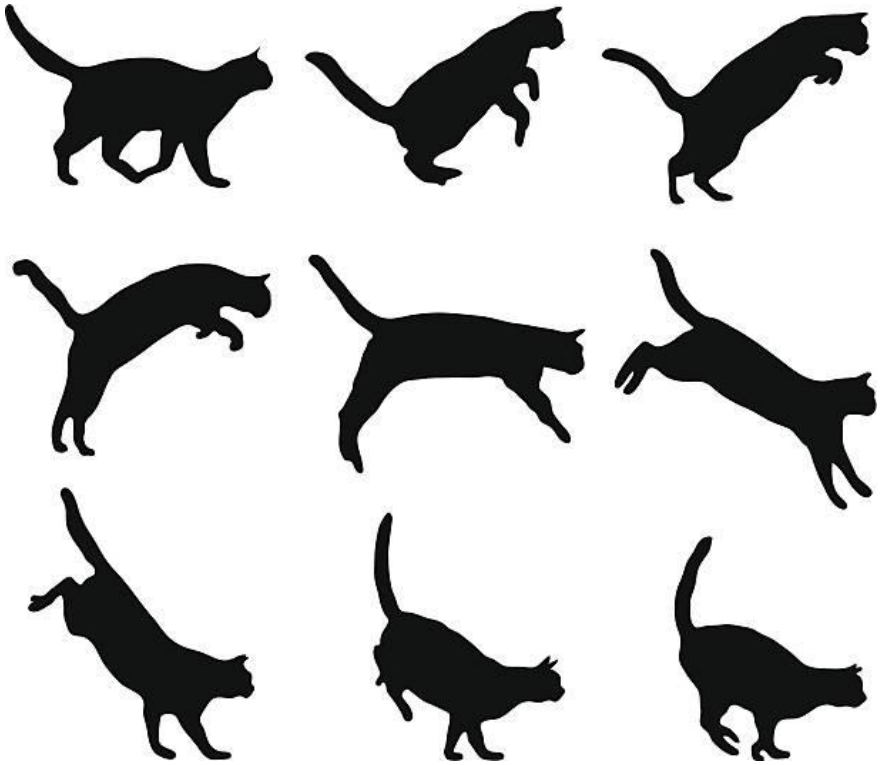

Figura 84: Plantilla de un gato saltando obtenida de: https://www.istockphoto.com/vector/cat-jumping-gm515811843-47552612

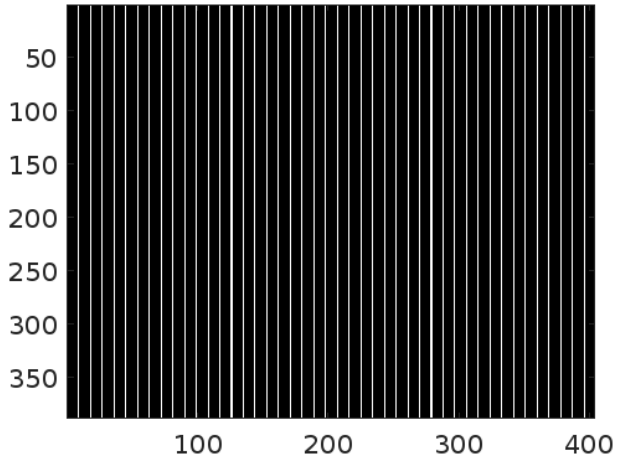

Figura 85: Rejilla cuadrada con franjas negras de 8 pixeles de ancho y franjas blancas de 1 píxel de ancho.

Una vez separadas las imágenes, se deben franjear de manera que concuerden con la rejilla cuadrada que se usará, como se muestra en las Figuras 86 a 94.

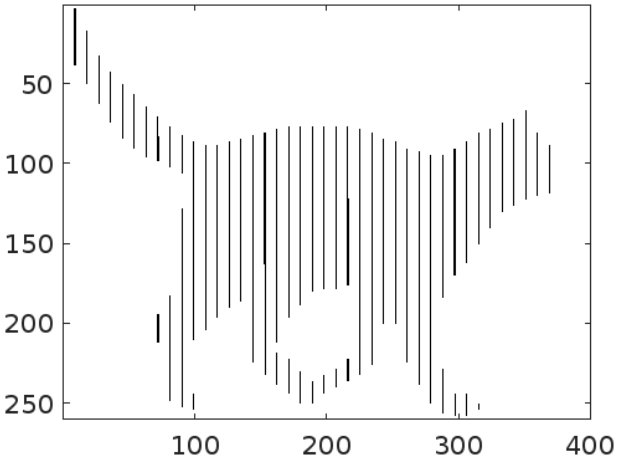

Figura 86: Primera imagen franjeada

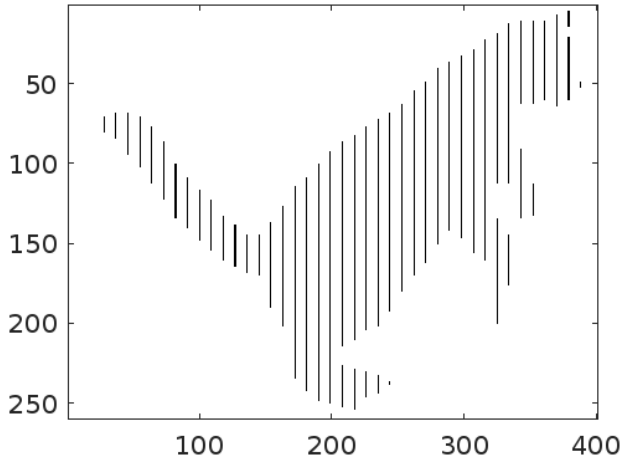

Figura 87: Segunda imagen franjeada

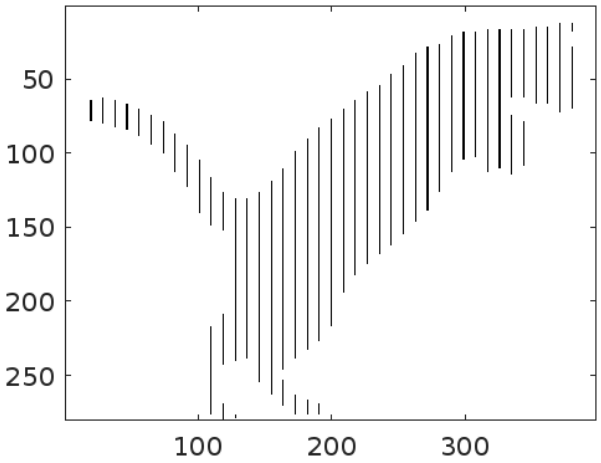

Figura 88: Tercera imagen franjeada

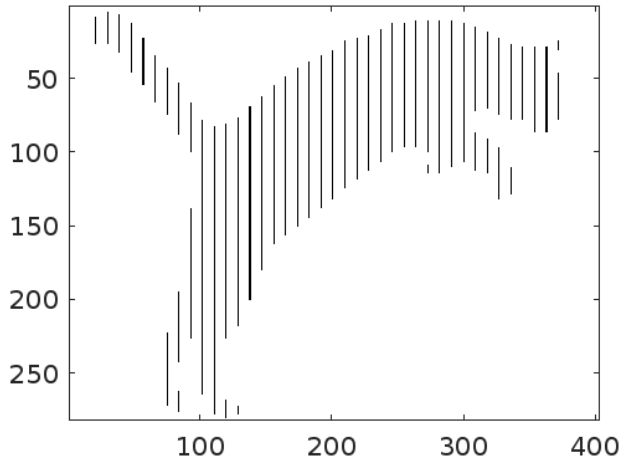

Figura 89: Cuarta imagen franjeada

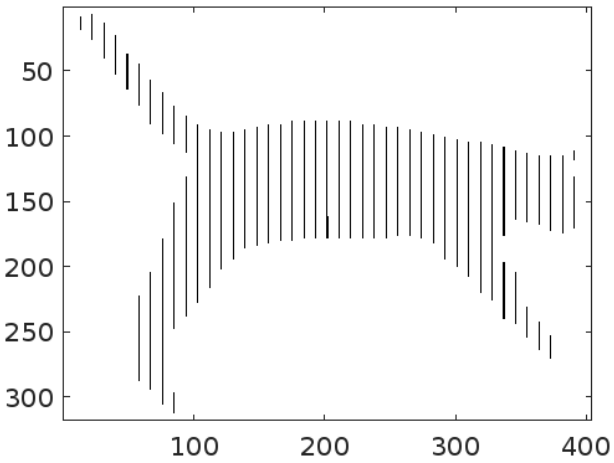

Figura 90: Quinta imagen franjeada

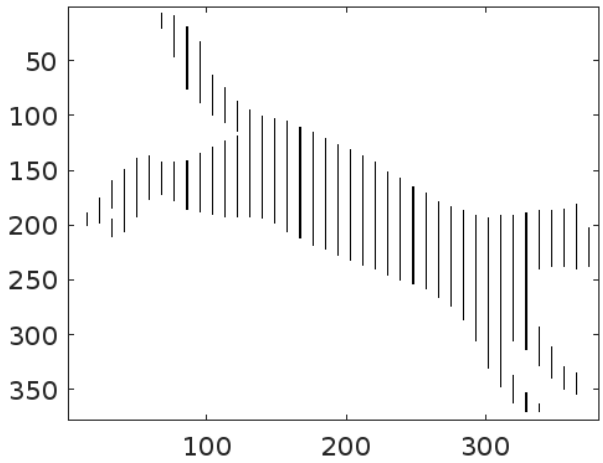

Figura 91: Sexta imagen franjeada

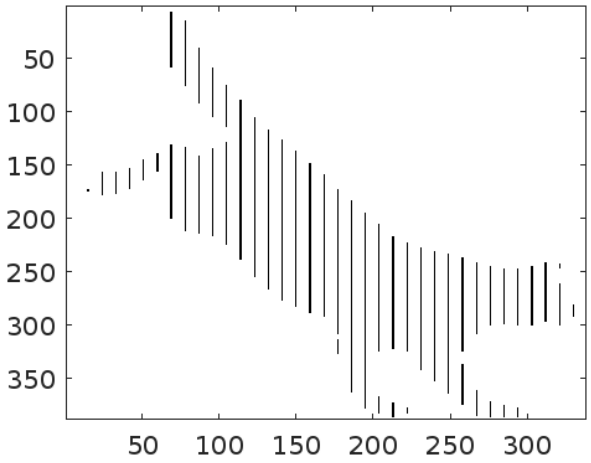

Figura 92: Séptima imagen franjeada

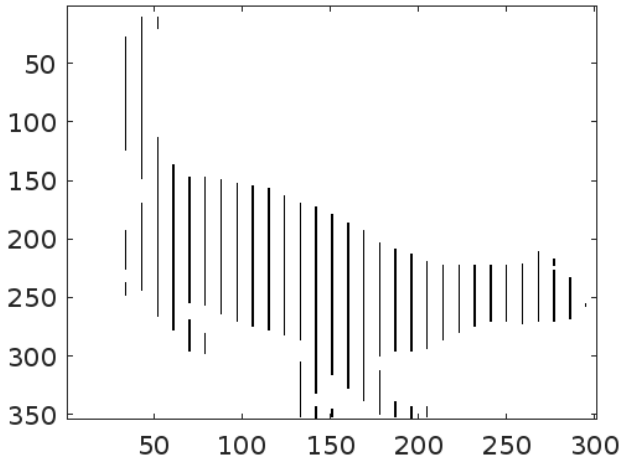

Figura 93: Octava imagen franjeada

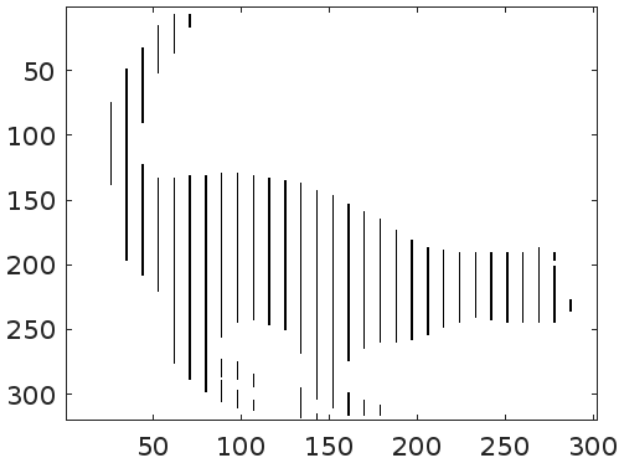

Figura 94: Novena imagen franjeada

Nuevamente, para que todos los pasos queden dentro de una imagen, se acomodan las rejillas individuales consecutivamente, con cada uno un píxel a la derecha que la imagen franjeada anterior sin que se estorben unas a otras y dando como resultado la rejilla de la Figura 95.

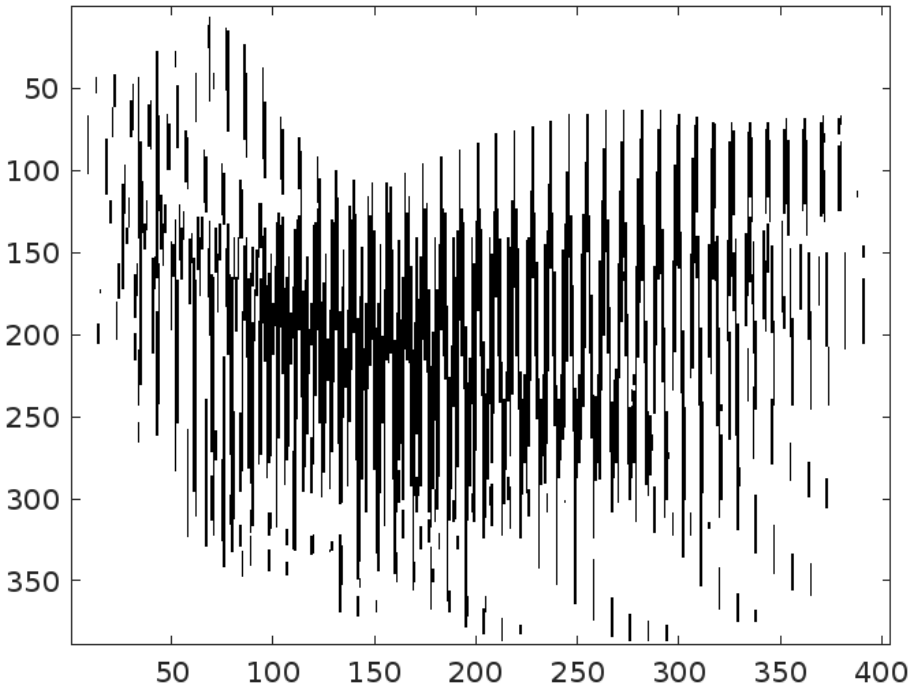

Figura 95: Rejilla completa del gato saltando

Finalmente, superponiendo la rejilla cuadrada de Figura 85 por encima de la rejilla completa del gato saltando, moviendo la rejilla cuadrada con pasos de 1 píxel a la derecha, se obtienen las imágenes originales, como se muestra en las Figuras 96 a 104.

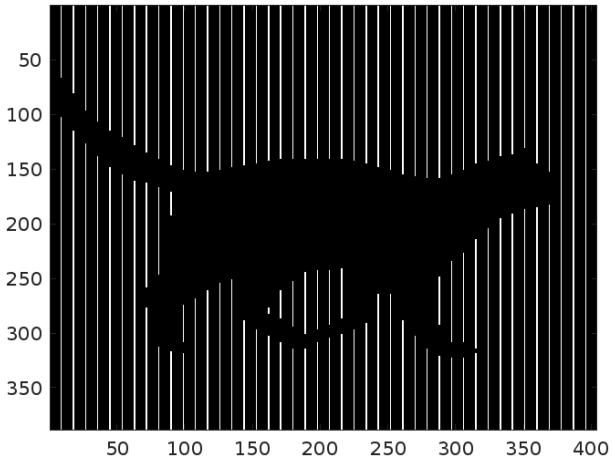

Figura 96: Primer paso del salto

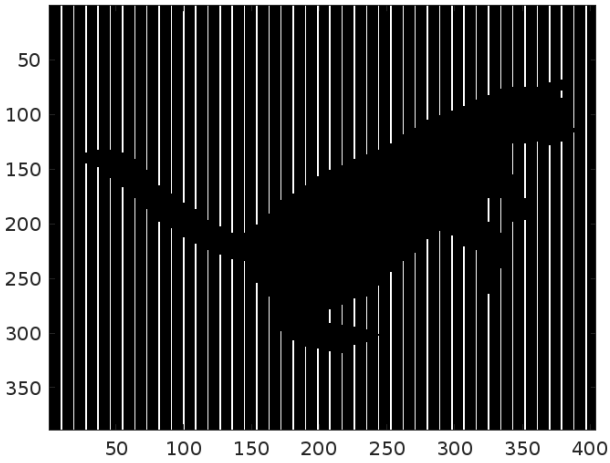

Figura 97: Segundo paso del salto

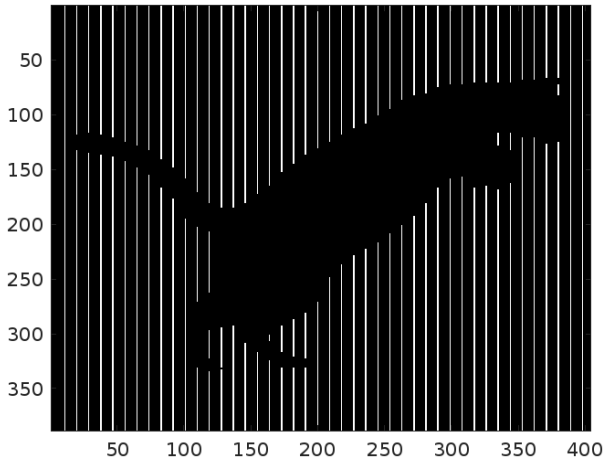

Figura 98: Tercer paso del salto

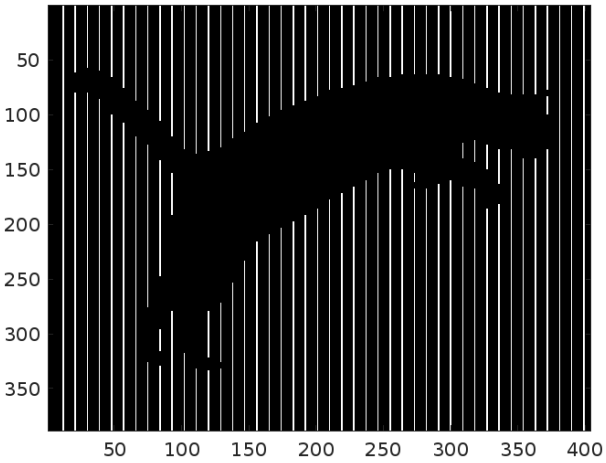

Figura 99: Cuarto paso del salto

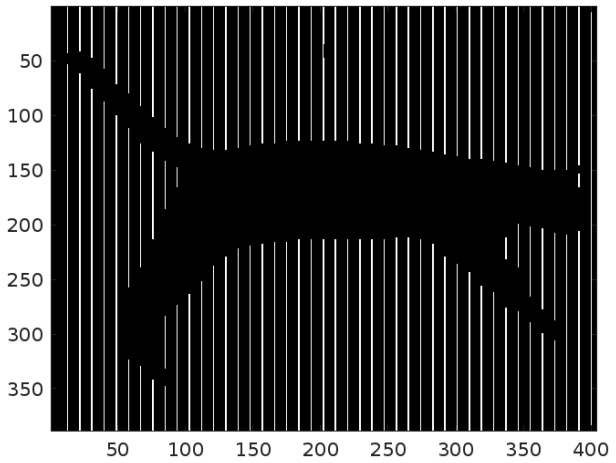

Figura 100: Quinto paso del salto

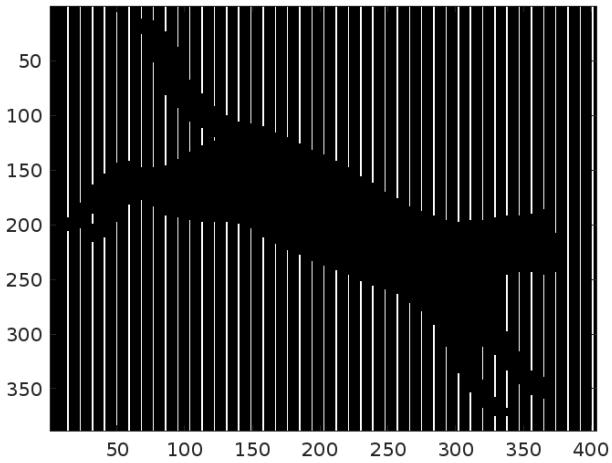

Figura 101: Sexto paso del salto

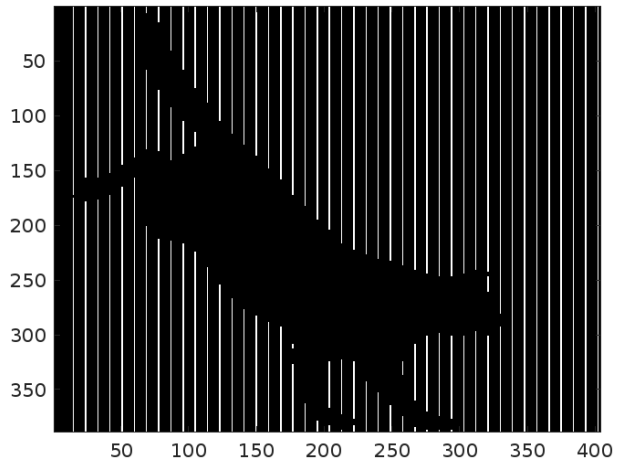

Figura 102: Séptimo paso del salto

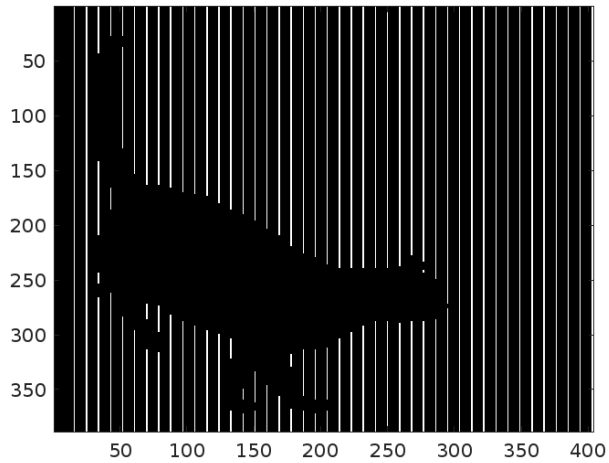

Figura 103: Octavo paso del salto

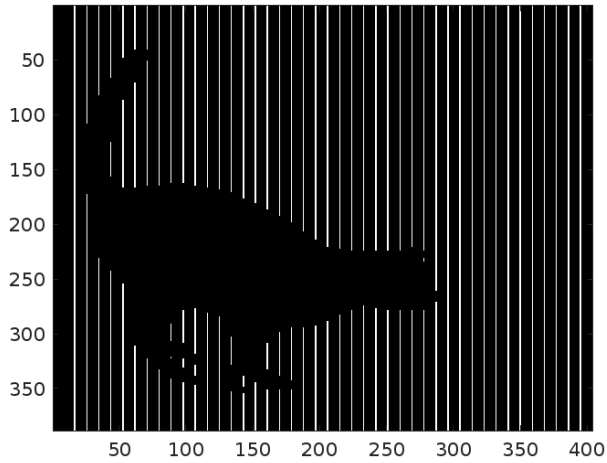

Figura 104: Noveno paso del salto

En condiciones reales, la secuencia de imágenes produce la ilusión de que el gato se encuentra en movimiento. Cualquier otra secuencia de imágenes tendría un resultado similar, y de esta forma se ha obtenido una manera de hacer animaciones en papel ayudándose del fenómeno Moiré. Es importante notar aquí, que se puede hacer una animación más fluida con más imágenes individuales, incluyendo pasos intermedios, pero con N imágenes de la animación, la rejilla cuadrada tendría franjas negras de $N-1$ pixeles de ancho con franjas claras (blancas) de 1 pixel de ancho, entonces con valores de $N > 10$ la imagen final es muy oscuro y difícil de ver. Por esto se recomienda usar $N < 10$ imágenes individuales.

7.1 Código *MatLab* para el gato saltando

```
%------------------------ANIMACION GATO------------------
% Para hacer animacion con Moire buscamos formar figuras al mover una
% rejilla de onda cuadrada por lo que cada frame debe acompletarse con
% dicha rejilla. Le quitaremos franjas a las imagenes originales con un
% desfase de un pixel y luego combinaremos las imagenes para
% hacer una sola rejilla y al ponerle una rejilla cuadrada se formara la
% imagen original y al moverla hacia abajo aparecerá el siguiente paso,
% esta animacion producira la ilusion de un gato saltando.

% El usuario debe bajar la imagen completo, la Figura 84, de la liga:
%  https://www.istockphoto.com/vector/cat-jumping-gm515811843-47552612
% Debe darle el nombre “cat2.jpg”, y asegurar que está en un path que
% ser leída por este programa.



% Leemos las imagen a utilizar para la animacion y las ponemos en formato
% blanco y negro.
G=double(im2bw(imread('cat2.jpg'),0.4));

% Ahora conseguimos las imagenes individuales de cada paso
A{1}=imcrop(G,[1,1,199,129]);
A{2}=imcrop(G,[200,1,199,129]);
A{3}=imcrop(G,[415,1,199,139]);
A{4}=imcrop(G,[1,178,199,140]);
A{5}=imcrop(G,[211,174,199,158]);
A{6}=imcrop(G,[425,170,199,188]);
A{7}=imcrop(G,[13,338,165,193]);
A{8}=imcrop(G,[250,355,146,176]);
A{9}=imcrop(G,[465,372,146,159]);

Asize=size(A);
```

```
noIm=Asize(1,2); % Este parametro guarda el numero de imagenes que hay

% Agrandamos las imagenes al doble para que la imagen este mas clara
for i=1:noIm
   A{i}=im2bw(imresize(A{i},2),0.4);
end

% IMPORTANTE: Rotamos las imagenes para que podamos al final la rejilla se
% mueva en la misma direccion en que camina el gato y produzca la animacion
% si usted no desea girar las imagenes comente el siguiente for y mas
% adelante tambien comente cuando se regresa la imagen a su orientacion
% original y comente for para rotar la rejillas cuadradas.

for i=1:noIm
   A{i}=imrotate(A{i},-90);
end

% Con imgrid generamos las franjas en las imagenes, y con step agregamos
% una fila blanca al inicio de cada numero para que al combinar las
% imagenes las lineas de cada numero no se estorben entre si.
for i=1:noIm
   AG{i}=imgrid(A{i},noIm-1,1);
   AG{i}=step(AG{i},(i-1));
end
% Pedimos ver un ejemplo
figure
showImage(imrotate(AG{1},90))


% La funcion alignmat va a alinear las matrices que representan las
% imagenes y va a formar una matriz que tenga la informacion de las
% anteriores. El for empieza en 2 porque la primera imagen es la base.
AN=AG{1};
for i=2:noIm
   AN=alignmat(AG{i},AN);
end

% IMPORTANTE: Si no roto las imagenes al inicio, comente la siguiente linea
% pues este comando regresa las imagenes rotadas a su orientacion original.
AN=imrotate(AN,90);

% Pedimos ver la rejilla final con toda la informacion de las anteriores
figure
showImage(AN)

% Guardamos las medidas de la rejilla final AN para utilizarlo mas abajo.
sAN=size(AN);

% Generamos la malla que acompletara cada uno de los numeros
rc=vargrid(noIm-1,1,sAN(1,1),sAN(1,2));
src=size(rc); % Guardamos sus dimensiones
% Pedimos una visualización de la rejilla
figure
showImage(rc)
```

```
% Ahora hagamos el desplazamiento de la rejilla anterior desde el espacio
% de frecuencias para esto consideremos que un desplazamiento se ve como un
% cambio de fase: A*e^(i*2*pi*u/N)

% Declaramos la Transformada de las rejilla cuadrada original y la rejilla
% de numeros,Trc y TAN respectivamente.
T_rc{1}=fftshift(fft2(rc)/(src(1,1)*src(1,2)));
T_AN=fftshift(fft2(AN)/(sAN(1,1)*sAN(1,2)));

% IMPORTANTE: Defina el movimiento en pixeles que desea, nótese que puede
% pedir movimientos de menos de un pixel, p.e. 1/4 de pixel
dpx=1;

% Notese que si dpx es diferente de 1, entonces aumenta o dismnuye el
% numero de pasos que necesitamos para visualizar toda la rejilla, si es
% menor a 1 necesitaremos mas pasos, si es mayor a 1 entonces necesitaremos
% menos pasos.
until=noIm*(1./dpx);

% El for empieza en 2 porque no hace falta mover la rejilla para el primer
% numero y aqui se desplazan las rejillas.

for i=2:1:until
   for j=1:sAN(1,2)
       for k=1:sAN(1,1)
           T_rc{i}(j,k)=T_rc{1}(j,k)*exp(-1i*2.*pi*j*(dpx)*(i-1)/sAN(1,2));
       end
   end
end

% IMPORTANTE: Comente el siguiente for si eligio no rotar las imagenes
% originales
for i=1:until
    T_rc{i}=imrotate(T_rc{i},90);
end


% En este for se convolucionan la rejilla de numeros con las rejillas de
% franjas desplazadas.
for i=1:1:until
   Conv_TAN=convolve2(T_AN,T_rc{i},'circular');
   TI=abs(ifft2(Conv_TAN));
   fig=figure;
   showImage(TI)

   %
   %-------OPCIONAL: GIF ----------------------------------------
   % Ponga una { en el % libre de arriba si no desea que se guarde el GIF
   % Ahora hacemos un gif con las imágenes obtenidas de los números
   frameo = getframe(fig);
   im = frame2im(frameo);
   [imind,cm] = rgb2ind(im,256);
   outfile = 'gato.gif';
```

```
   % On the first loop, create the file. In subsequent loops, append.
   if i==1
       imwrite(imind,cm,outfile,'gif','DelayTime',0,'loopcount',inf);
   else
       imwrite(imind,cm,outfile,'gif','DelayTime',0,'writemode','append');
   end
   %}
end


function [imgd] = imgrid(mat,b,w)
% Funcion que borra renglones de una imagen mat, los renglones son de ancho
% w pixeles y estan espaciados por b pixeles.
   s=size(mat);
   imgd=mat;
   j=1;
   bw=b+w;
% Checamos que los parámetros b y w no excedan las dimensiones de mat
   if b>s(1,1) || w>s(1,1)
       fprintf('check size')
% En caso de que los parametros sean viables, procedemos con la funcion
   else
       for i=0:s(1,1)      % Nos ubicamos en la 1ra entrada de la matriz
           j=i*bw+1;       % Esto nos ubica en donde iniciara el renglon
           if j>s(1,1)     % Verificamos que seguimos dentro de la matriz
               break       % En caso contrario acabamos el proceso
          end
          while j<=b+(i*bw) % Nos mantiene dentro del renglon que borraremos
               if j>s(1,1)
                   break   % Verificamos que seguimos dentro de la matriz
               end
               imgd(j,:)=1;     % Aquí borramos todo el renglon actual
               j=j+1;           % Pasamos al siguiente renglon de la matriz
           end
       end
   end
end

function [grid]=vargrid(b,w,xpix,ypix)
% Esta funcion hace rejillas cuadradas con renglones negros de b pixeles de
% ancho y renglones blancos de w pixeles de ancho. La rejilla tiene xpix
% columnas y ypix renglones.
   grid=ones(ypix,xpix);  % Definimos una matriz inicial de 1's
% Y con el mismo codigo de la funcion imgrid, le hacemos los renglones
% negros
   j=1;
   bw=b+w;
   if b>ypix || w>ypix
       fprintf('check size')
   else
       for i=0:ypix
           j=i*bw+1;
           if j>ypix
               break
```

```
            end
            while j<=b+(i*bw)
                if j>ypix
                    break
                end
                grid(j,:)=0;
                j=j+1;
            end
        end
    end
end

function [mats]=step(mat,pix)
% Esta funcion agrega renglones blancos al inicio de una matriz mat, el
% renglon tiene ancho pix pixeles.
    s=size(mat);
    mat0=ones(pix,s(1,2)); % Hacemos una matriz de 1's con el ancho deseado
    mats=vertcat(mat0,mat); % La colocamos arriba de la matriz mat
end

function [amat]=alignmat(mat1,mat2)
% Esta funcion combina dos matrices mat1 y mat2 desde sus centros esta
% funcion esta pensada para combinar matrices que representen imagenes a
% b&w cuya informacion no se superpone, esto es que en una misma celda solo
% una de las matrices puede ser negra (0).
% La idea para combinar las matrices es sumar los valores de una misma
% celda en cada matriz para obtener el de la matriz combinada, pero como
% blanco mas blanco debe ser blanco, y esto es 1 + 1 = 1, entonces mejor
% intercambiamos los 1 por 0 y viceversa, de esta forma los pixeles blancos
% ahora son 0 y 0+0 = 0 y en caso de que haya negro en una imagen y en la
% otra blanco, tendriamos 1+0=1 (negro)
    mat1=double(~mat1);
    mat2=double(~mat2);
    s1=size(mat1);
    s2=size(mat2);
    dsy=s1(1,1)-s2(1,1);  % Las matrices pueden tener distinto tamaño por lo
    dsx=s1(1,2)-s2(1,2);  % que guardamos el valor de esas diferencias.
    if dsy==0 & dsx==0    % Si tienen la misma dimension, la combinacion es
        amat=mat1+mat2;   % la suma directa de ambas matrices
    else
        m=1;
        n=1;
        if dsy<0          % Si son de distinto tamaño, localizamos la matriz
            m=s2(1,1);    % con mas renglones
        elseif dsy>=0
            m=s1(1,1);
        end
        if dsx<=0         % Localizamos la matriz con más columnas
            n=s2(1,2);
        elseif  dsx>0
            n=s1(1,2);
        end
        % Definimos la matriz que tendrá los valores combinados
        amat=zeros(m,n);
```

```
    % La alineación es en el centro pero el calculo del centro de una
    % matriz cambia si las dimensiones son pares o impares, cx1 guarda
    % la coordenada x del centro de mat1, cx2 la coordenada x del centro
    % de mat2. Notese que solo alinearemos con el centro en x porque de
    % acuerdo al objetivo del programa nos interesa que en y no haya
    % desplazamiento alguno, por lo que acomodaremos desde el primer
    % renglon
     if rem(s1(1,2),2) == 0
        cx1=(s1(1,2)/2)+1;
    elseif rem(s1(1,2),2) == 1
        cx1=ceil(s1(1,2)/2);
    end
    if rem(s2(1,2),2) == 0
        cx2=(s2(1,2)/2)+1;
    elseif rem(s2(1,2),2) == 1
        cx2=ceil(s2(1,2)/2);
    end
    % También localizamos el centro de la matriz combinada amat
    if rem(n,2) == 0
        cxm=(n/2)+1;
    elseif rem(n,2) == 1
        cxm=ceil(n/2);
    end
    % Este for ingresa los valores de mat1 directo en amat pero los
    % coloca de tal forma que el centro de mat1 quede en el centro de
    % amat, de acuerdo al objetivo de esta funcion es importante que en
    % y no se desplacen las matrices por lo que los valores de mat1 se
    % ingresan desde el primer renglon.
    for i=1:s1(1,1)
        for j=1:s1(1,2)
            amat(i,(cxm-cx1)+j)=mat1(i,j);
        end
    end
    % Ahora se mete la información de mat2 a amat, pero no debemos
    % borrar la información ya existente de mat1. Notese que mat2
    % tambien se ingresa de forma centrada.
    for i=1:s2(1,1)
        for j=1:s2(1,2)
            % Si estamos fuera de las celdas que ya tienen info de mat1
            % entonces ponemos directo la info de mat2
            if (cxm-cx2)+j<(cxm-cx1)+1 || (cxm-cx2)+j>(cxm-cx1)+s1(1,2)
                amat(i,(cxm-cx2)+j)=mat2(i,j);
            else
                if i>s1(1,1)
                    amat(i,(cxm-cx2)+j)=mat2(i,j);
            % Si estamos en celdas con info de mat1, entonces sumamos el
            % valor preexistente de mat1 con el nuevo de mat2
                else
                    amat(i,(cxm-cx2)+j)=amat(i,(cxm-cx2)+j)+mat2(i,j);
                end
            end
        end
    end
end
```

```
   amat=double(~amat); % Regresamos a la escala de colores original
end

function [c1d] = Xcomb1dp(pts,sep,dx)
% funcion para definir un arreglo con la digitalizacion de la funcion
% comb() en 1 dimension. Hay una delta() en el punto central (pts/2)+1.
% pts es el numero de puntos totales en el arreglo,
% y debe ser una potencia de 2: 2,4,8,16,32,64,128,.....
% sep es la separacion de cada delta() en la comb()
c1d = zeros(1,pts); % crear arreglo para la funcion delta()
c1d(pts/2+1+dx) = 1;  % el punto central es una delta()
% agregar las otras deltas al arreglo
for i = 1:ceil(pts/(2*sep))+1+dx
   if (pts/2+1+dx+(i*sep)) <= pts
       c1d(pts/2+1+dx+(i*sep)) = 1;
   end
   if (pts/2+1+dx-(i*sep)) >= 1
       c1d(pts/2+1+dx-(i*sep)) = 1;
   end
end
end

function [c2d] = Xcomb2dp(pts,sepx,sepy,dx,dy)
% funcion para definir un arreglo con la digitalizacipn de la funcion
% comb() en 2 dimensiones. Hay una delta() en el punto central.
% pts es el numero de puntos totales en el arreglo,
% y debe ser una potencia de 2: 2,4,8,16,32,64,128,.....
% sepx es la separacion de cada delta() en la comb() en x
% sepy es la separacion de cada delta() en la comb() en y
cx1d = Xcomb1dp(pts,sepx,dx);   % 1d comb() in x
cy1d = Xcomb1dp(pts,sepy,dy);   % 1d comb() in y
% generar una matriz pts*pts con solo una fila = cx1d
cx2d = zeros(pts,pts);
cx2d(pts/2+1+dx,:) = cx1d;
% generar una matriz pts*pts con solo una columna = cy1d
cy2d = zeros(pts,pts);
cy2d(:,pts/2+1+dy) = cy1d.';
% la convolucion en 2d de cx2d con cy2d da la matriz de la comb() en 2d
c2d = conv2(cx2d,cy2d,'full');
end

function [r2d] = Xrect2d(pts,widx,widy)
% funcion para definir un arreglo con la digitalizacion de la funcion
% rect() en 2 dimensiones.
% pts*pts es el numero de puntos totales en el arreglo,
% y pts debe ser una potencia de 2: 2,4,8,16,32,64,128,.....
% widx es el numero de puntos en la direccion x con valor 1 y debe
% ser entre 1 y pts
% widy es el numero de puntos en la direccion y con valor 1 y debe
% ser entre 1 y pts
% si widx o widy no es menor que pts, obligar a que wid = pts-2
if (widx > pts)||(widx < 1)
   widx = pts - 2;
end
```

```
if (widy > pts)||(widy < 1)
   widy = pts - 2;
end
x = 1:pts;
y = 1:pts;
[xr2d,yr2d] = meshgrid(x,y);   % crear matriz de valores x,y para la funcin rect()
minptsx = pts/2 - ceil(widx/2) + 1;    % posicion en la matriz donde empieza valores
= 1
minptsy = pts/2 - ceil(widy/2) + 1;
                            % agregar 1s en el centro de la matriz
r2d = ((xr2d >= minptsx) & (xr2d < minptsx + widx) & (yr2d >= minptsy) & (yr2d <
minptsy + widy));
end

function showImage(img,x,y)
   if ~exist('x','var')
       x = 1:size(img,2);
   end
   if ndims(x)==2
       x = x(1,:);
   end
   if ~exist('y','var')
       y = 1:size(img,1);
   end
   if ndims(y)==2
       y = y(:,1)';
   end

   imagesc(x,y,img)
%    axis equal
%    axis tight
   colormap(gray);
end
```

8. Código *MatLab* para la función convolve2( )

El código para la función convolve2( ) utilizado en los programas de MatLab presentados aquí está disponible en el “File Exchange” de MatLab en la liga (Young, 2025):

https://www.mathworks.com/matlabcentral/fileexchange/22619-fast-2-d-convolution

9. Conclusiones

Se estudiaron los patrones de Moiré, que son patrones resultantes de la superposición de dos o más rejillas, desde una perspectiva del análisis de Fourier. Se entendió la superposición de rejillas como una multiplicación de funciones de reflectancias y, por ende, se llegó a que la superposición de rejillas está dada por la convolución de espectros de éstas en el espacio de Fourier. Este aspecto del trabajo podría ser útil como una demostración en un clase de Fourier.

También se presentó la producción y manipulación de patrones a través de la superposición de rejillas periódicas como: rejillas cosinusoidales, cuadradas, de puntos y de anillos, esto con ayuda de programas hechos en Matlab.

Finalmente, se presentaron otros programas en Matlab que permitieron producir animaciones Moiré arbitrarios que cambian con la posición relativa entre las rejillas. El primero fue una secuencia de números del 0 al 9, en la cual se mostró resolución subpíxel, y que se podría utilizar para realizar mediciones en un vernier de Moiré. El segundo fue una secuencia que formaba la ilusión de un gato saltando, este programa podría emplearse para crear animaciones llamativas para el público en general.

10. Referencias Bibliográficas

Amidror, I. (2009), *The Theory of the Moiré Phenomenon*, London, Springer.

Bergkvist, L. A. and Forsen, I. (1986), *Leading mark indicator*, Patente de los Estados Unidos, US4629325A.

Goodman, J.W., 1996, *Introduction to Fourier Optics, Second Edition*, Boston, McGraw-Hill, Capítulo 2

Gustafsson, M. (2000) “Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy”, Journal of Microscopy, 198(2), pp. 82–87.

Kamal, H., Völkel, R., y Alda, J. (1998), “Properties of Moiré magnifiers”, Optical Engineering, 37(11), pp. 3007–3014.

Kazda, A. and Caves, R. (eds.) (2015), *Airport design and operation*, Leeds, UK, Emerald Group Publishing Limited.

Kobayashi, K. (2000) “Moiré pattern in scanning tunneling microscopy”, Physical Review B, 53(16), pp. 11091–11099.

MacDonald, A. H. (2019), “Bilayer graphene’s wicked, twisted road”, Physics Online Journal, 12, 12.

Miao, H., Panna, A., Gomella, A. A., Bennett, E. E., Znati, S., Chen, L., and Wen, H. (2016), “A universal Moiré effect and application in x-ray phase-contrast imaging”, Nature Physics, 12, pp. 830–834.

Reid, G. (1984), “Moiré fringes in metrology”, Optics and Lasers in Engineering, 5(2), pp. 63–93.

Schouteden, K., Galvanetto, N., Wang, C., Li, Z., and Haesendonck, C. V. (2015), “Scanning probe microscopy study of chemical vapor deposition grown graphene transferred to Au(111)”, Carbon, 95, pp. 318–322.

Seder, R. B. (2024), *Pequeño Gato*, México, Novelty Ediciones.

Stevens, R. (1999), “Optical inspection of periodic structures using lens arrays and Moiré Magnification”, The Imaging Science Journal, 47(4), pp. 173–179.

Young D. (2025), “Fast 2-D Convolution”, MATLAB Central File Exchange, https://www.mathworks.com/matlabcentral/fileexchange/22619-fast-2-d-convolution recuperado 2 abril, 2025.