

Procesamiento de Imagen y señal

Trabajo de Procesamiento de Imagen

Javier Aguirre

14 de Febrero de 2022

Contents

Introducción al Procesamiento de Imagen	3
Ejercicio 2a	4
Definición del filtro	4
Ejecución del filtrado	4
Construcción la nueva imagen	5
Ejercicio 2b	7
Funciones básicas	7
Reproducción de la imagen de dolár	7
Información sobre la imagen	7
Mostramos la imagen original	7
Ampliación del rango de valores de intensidad	9
Separación de la imagen en diferentes planos	9
Comparación de ambas imágenes	13
Ejercicio 3	17
Quitar los semitonos de la imagen, haciéndola más uniforme	17
Lectura de la imagen	18
Aplicación de la Transformada de Fourier	18
Diseño de la mascara para el filtrado	20
Aplicación de la máscara	22
Transformada inversa	23

Ejercicio 4	26
Binarización de la imagen	26
Esqueletonizar	27
Conclusiones y dificultades	29

Introducción al Procesamiento de Imagen

El procesamiento de imágenes digitales es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información.

Es el conjunto de técnicas englobadas dentro del preprocesamiento de imágenes cuyo objetivo fundamental es obtener, a partir de una imagen origen, otra final cuyo resultado sea más adecuado para una aplicación específica mejorando ciertas características de la misma que posibilite efectuar operaciones del procesado sobre ella.

Los principales objetivos que se persiguen con la aplicación de filtros son:

-Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.

-Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.

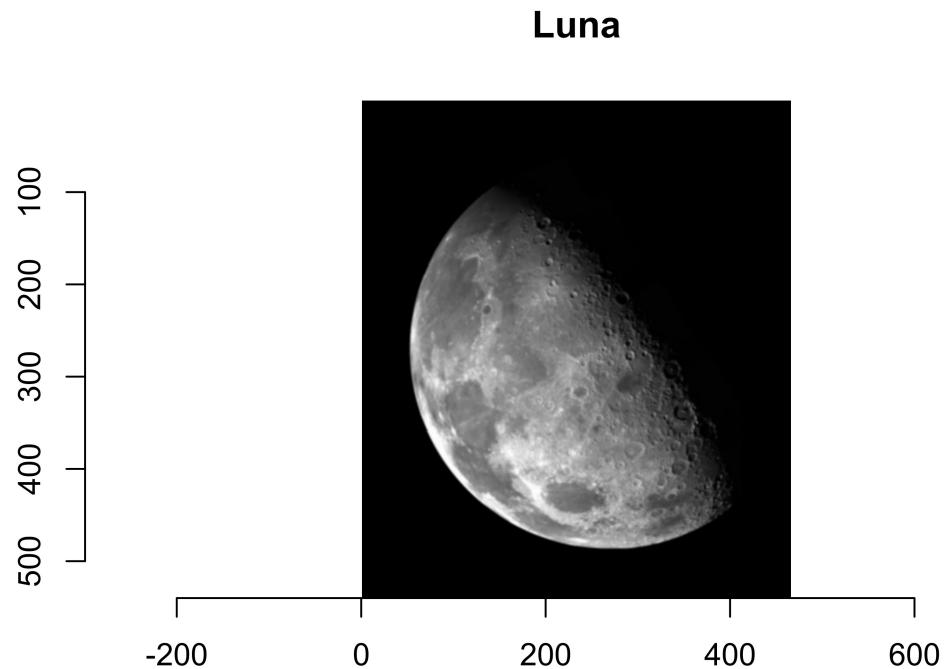
-Realzar bordes: destacar los bordes que se localizan en una imagen.

-Detectar bordes: detectar los píxeles donde se produce un cambio brusco en la función intensidad.

Ejercicio 2a

Cargaremos a continuación la imagen de la luna.

```
library("imager")
library("magick")
fImagen <- 'Fig0338(a)(blurry_moon).jpg'
imag <- load.image(fImagen)
plot(imag, main="Luna")
```



Definición del filtro

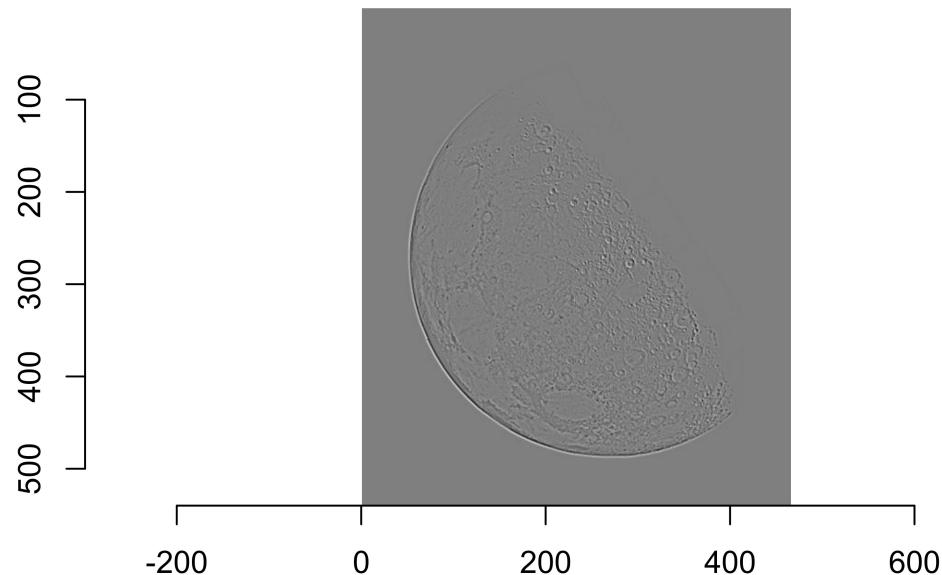
Para este ejercicio utilizaremos el filtro de Laplace. Este filtro detecta bordes en la imagen usando el método laplaciano, que produce bordes finos de un píxel de ancho.

```
filtrado <- function(imagen){
  return(imlap(imagen))
}
```

Ejecución del filtrado

```
laplaceFilter <- filtrado(imag)
laplaceFilter.min <- laplaceFilter-min(laplaceFilter)
laplaceFilter <- (laplaceFilter.min/max(laplaceFilter.min))
plot(laplaceFilter, main = "Filtro de laplace")
```

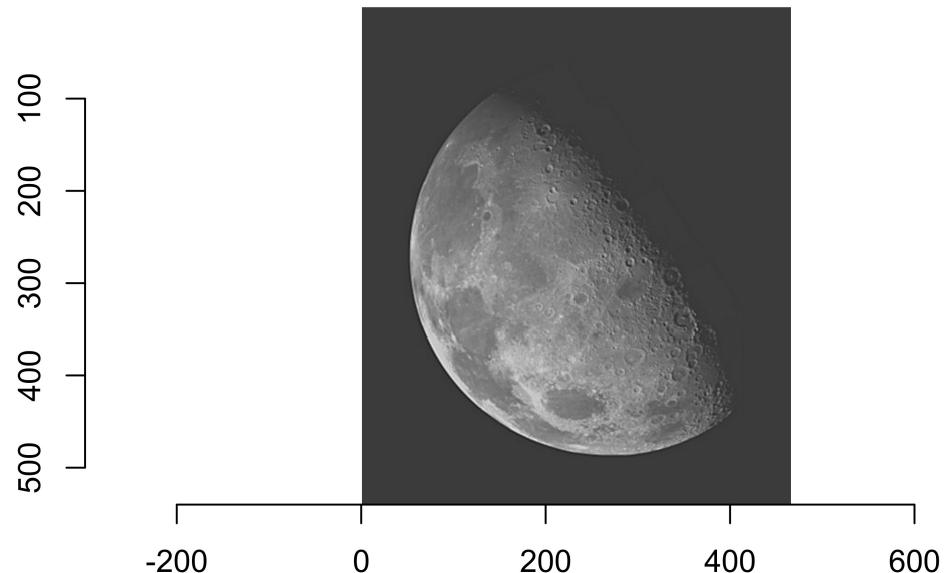
Filtro de laplace



Construcción la nueva imagen

```
imagenFinal <- imag - laplaceFilter
imagenFinal.min <- imagenFinal-min(imagenFinal)
imagenFinal.max <- (imagenFinal.min/max(imagenFinal.min))
plot(imagenFinal.max, main = "Imagen Filtrada")
```

Imagen Filtrada



Ejercicio 2b

Funciones básicas

Reproducción de la imagen de dolár

Cargamos Imager y magick y procedemos a cargar la imagen .tif de 100-dollars en el directorio correspondiente.

```
library("imager")
library("magick")
library(moments) # needed for finding the third and fourth moments
# Install if not already installed
library(pryr)
require(tiff)

WD <- getwd() # si la imagen está guardada en otro directorio
# debemos especificarlo
fimagen <- 'Fig0314(a)(100-dollars).tif'
#fimagen <- 'Fig0222(b)(cameraman).tif'
# nombre del fichero
fpath <- paste(WD,fimagen,sep = "/") # ruta de acceso

img <- readTIFF(fpath, native=TRUE) # leemos la imagen
img2 <- readTIFF(fpath) # leemos la imagen
#img2 <- load.image(fpath) # leemos la imagen
```

Información sobre la imagen

```
class(img)
```

```
## [1] "nativeRaster"
```

Mostramos la imagen original

Imprimimos en pantalla la imagen del dólar.

```
img2<-as.raster(img2)
plot(img2, main = "Figura 3.14 original.tif")
```



Ampliación del rango de valores de intensidad

Aplicación de la función al array de valores de intensidad.

```
# cimg -> matrix

cimg2matrix <- function(img){
  # img, imagen de tipo cimg
  # devuelve una matriz con las mismas dimensiones
  img.aux <- as.matrix(img)
  DM <- dim(img.aux)
  #img.mat <- img.aux[,DM[2]:1]
  img.mat <- img.aux
  return(img.mat)
}

img.matrix <- cimg2matrix(img) # matriz bidimensional
Img.array <- matrix(img.matrix,ncol=1,byrow = FALSE) # array unidimensional
```

Realizamos un aumento de contraste utilizando todo el rango de valores de salida.

```
# :[0,s.max]
piecew.contrast <- function(r,r.min,r.max,s.max){
  s <- ((r-r.min)/(r.max-r.min))*s.max
  return(s)
}

r.mx <- max(Img.array)
r.mn <- min(Img.array)
s.mx <- 2^(8)-1 # escala de valores [0,L-1 = 255]

#CSImg.array <- apply(Img.array,1,piecew.contrast,r.max=r.mx,r.min=r.mn,s.max = s.mx)
CSImg.array <- apply(Img.array,1,piecew.contrast,r.max=r.mx,r.min=r.mn,s.max = s.mx)
# aplicamos la función al array

#CSImg.array <- apply(Img.array,1,piecew.contrast,r.max=1,r.min=0,s.max = s.mx)# aplicamos la función a
object.size(CSImg.array) # tamaño en memoria del array

## 4768048 bytes
```

Separación de la imagen en diferentes planos

Vamos a separar la imagen en diferentes planos teniendo en cuenta los bits usados para guardar la imagen
Otra función de transformacion no monotona es la transformación por fraccionamiento en planos de bits
Descomponer una imagen en sus planos de bits para analizar la importancia relativa de cada bit en la imagen.

Un proceso que ayuda a determinar la idoneidad del número de bits usados en la cuantización de la imagen. También se utiliza en la compresión de imágenes.

```
object.size(img) # tamaño en memoria  
  
## 2384616 bytes  
  
length(img)  
  
## [1] 596000  
  
object_size(img) # tamaño en memoria MB  
  
## 2,384,616 B  
  
str(img)  
  
##  'nativeRaster' int [1:500, 1:1192] -16777216 -16777216 -16777216 -16777216 ...  
## - attr(*, "channels")= int 1
```

Extraemos todos los bits uno por uno desde el primero hasta el octavo (representados por c1...c8)

```
#cd <- as.double(Img.array)  
# str(cd)  
  
cd <- CSImg.array  
str(cd)  
  
## num [1:596000] 0 0 0 0 0 0 0 0 0 0 ...  
  
c1 = mod(cd, 2)  
c2 = mod(floor(cd/2), 2)  
c3 = mod(floor(cd/4), 2)  
c4 = mod(floor(cd/8), 2)  
c5 = mod(floor(cd/16), 2)  
c6 = mod(floor(cd/32), 2)  
c7 = mod(floor(cd/64), 2)  
c8 = mod(floor(cd/128), 2)
```

Combinamos la imagen de nuevo para formar el equivalente a la imagen original en escala de grises.

```
cc = (2 * (2 * (2 * (2 * (2 * (2 * c8 + c7) + c6) + c5) + c4) + c3) + c2) + c1)  
  
GreyImg.array <- apply(as.matrix(cc), 1, piecewise.contrast, r.max=r.mx, r.min=r.mn, s.max = s.mx)
```

Reconstrucción de las imágenes de los planos de bit.

```

m1 <- matrix(c1, ncol=500, byrow=FALSE)
m2 <- matrix(c2, ncol=500, byrow=FALSE)
m3 <- matrix(c3, ncol=500, byrow=FALSE)
m4 <- matrix(c4, ncol=500, byrow=FALSE)
m5 <- matrix(c5, ncol=500, byrow=FALSE)
m6 <- matrix(c6, ncol=500, byrow=FALSE)
m7 <- matrix(c7, ncol=500, byrow=FALSE)
m8 <- matrix(c8, ncol=500, byrow=FALSE)

im1 <- as.cimg(m1)
im2 <- as.cimg(m2)
im3 <- as.cimg(m3)
im4 <- as.cimg(m4)
im5 <- as.cimg(m5)
im6 <- as.cimg(m6)
im7 <- as.cimg(m7)
im8 <- as.cimg(m8)

```

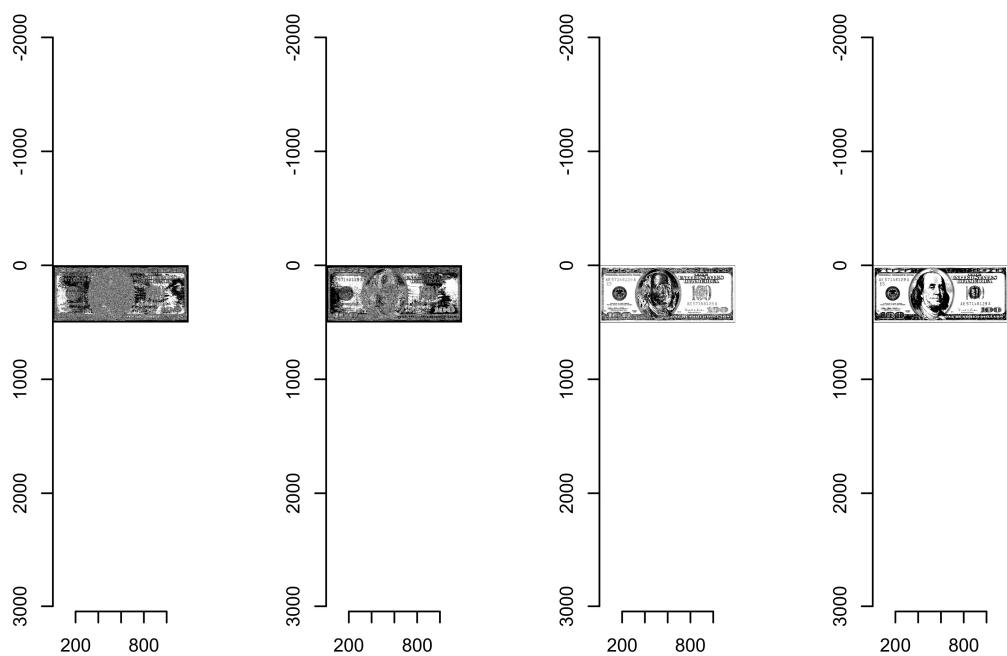
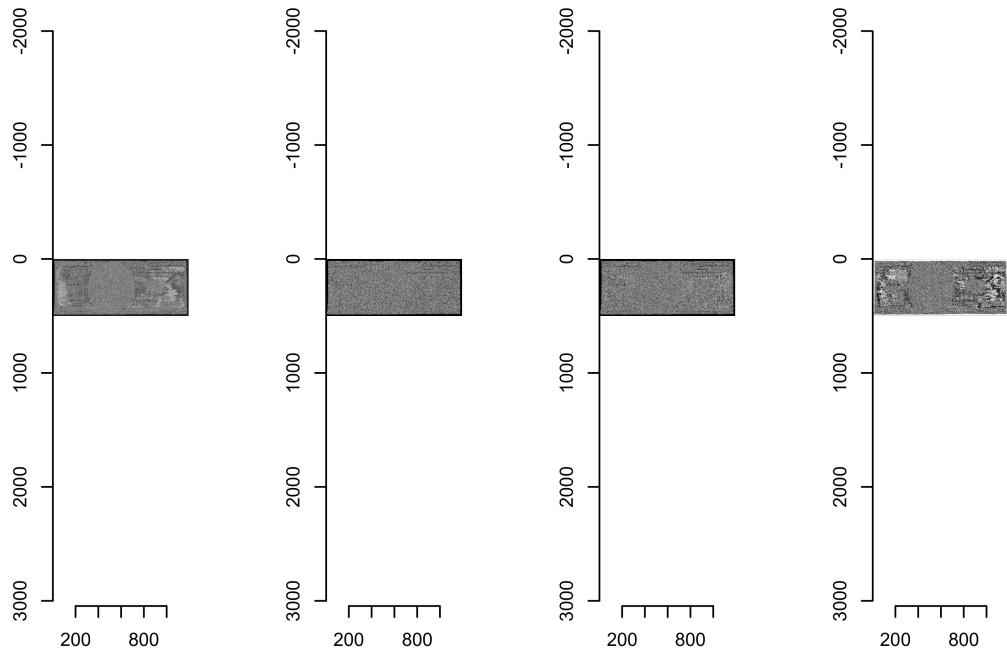
Mostramos los resultados obtenidos

```

layout(matrix(1:8,2,4,byrow = TRUE))

plot(im1)
plot(im2)
plot(im3)
plot(im4)
plot(im5)
plot(im6)
plot(im7)
plot(im8)

```

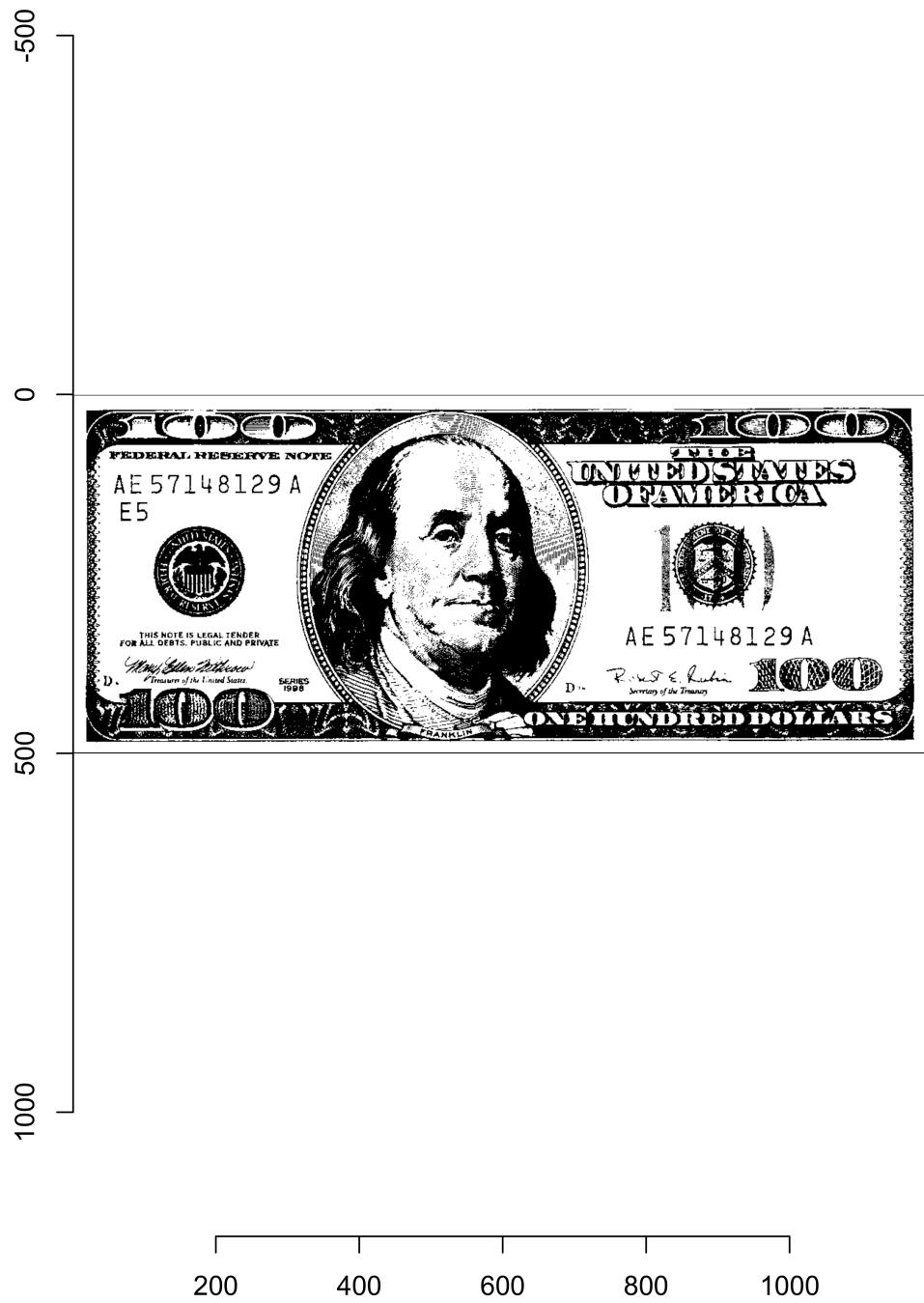


Comparación de ambas imágenes

```
plot(img2)
```



```
plot(im8)
```



Ejercicio 3

Quitar los semitonos de la imagen, haciendola más uniforme

Usaremos *imager* para mostrar las imágenes. *fftw* y *fftwtools* para calcular la transformada y el paquete *mmand* para hacer operaciones morfológicas.

```
# FUNCIONES AUXILIARES

espectro.imager <- function(fft.arg){
  # cimg
  Mod <- sqrt(fft.arg$real^(2)+fft.arg$imag^(2))
  return(Mod)
}

argumento.imager <- function(fft.arg){
  # cimg
  Arg.array <- atan2(as.matrix(fft.arg$real,nrow=1),as.matrix(fft.arg$imag,nrow=1))
  Arg <- as.cimg(matrix(Arg.array,ncol=dim(fft.arg$real)[2]))
  return(Arg)
}

# Pasamos la imagen al dominio de las frecuencias, vamos a modificar la imagen original fmn(-1)^(m+n)
Centra.frec <- function(M){
  M.aux <- M
  rows <- dim(M)[1]
  cols <- dim(M)[2]
  for (i in 1:rows){
    for (j in 1:cols) {
      M.aux[i,j] <- M[i,j]*(-1)^(i+j-2)
      # Trasparencia 21
    }
  }
  return(M.aux)
}

#####
#
# Filtro Butterword
#
#####

FButterword <- function (rows, cols, orden, radio){
  FB <- matrix(rep(0,rows*cols),rows,cols) #Inicializamos
  #n Orden del Filtro
  #DO Frecuencia Umbral
  for (i in 1:rows){
    for (j in 1:cols) {
      r <- sqrt((i-rows/2)^2+(j-cols/2)^2)
      FB[i,j] <- 1/(1+(r/radio)^(2*orden))
    }
  }
}
```

```
    return(FB)
}
```

Lectura de la imagen

```
HalfT <- imager::load.image('halftone.png')
```

Aplicación de la Transformada de Fourier

Antes de aplicar la transformada de Fourier vamos a cambiar la imagen al dominio de frecuencias. Veremos ahora las frecuencias en el centro de la imagen.

```
Matrix.Halft <- as.cimg(Centra.freq(as.matrix(HalfT)))
FFT.im <- fft(Matrix.Halft)

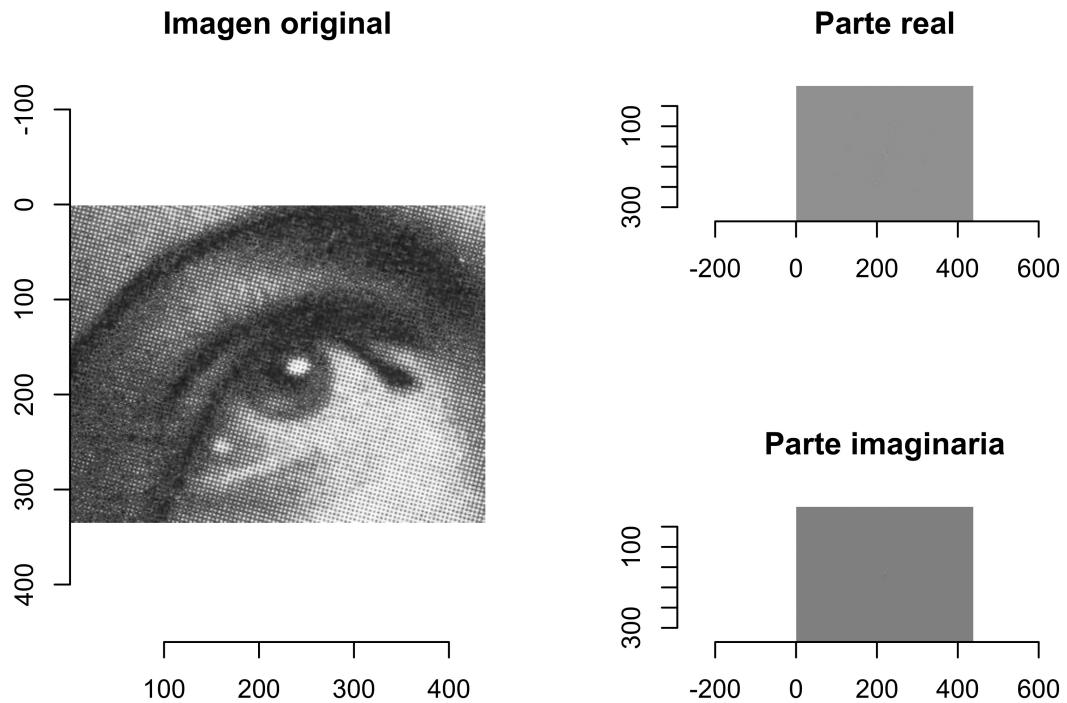
# FFT.im <- fft(HalfT)
```

Una vez pasada la imagen al dominio de frecuencias aplicamos la transformada.

```
# fft de la imagen centrada

layout(matrix(c(1,1,2,3),ncol =2))

plot(HalfT, main = "Imagen original")
plot(Re(FFT.im), main = "Parte real")
plot(Im(FFT.im), main = "Parte imaginaria")
```

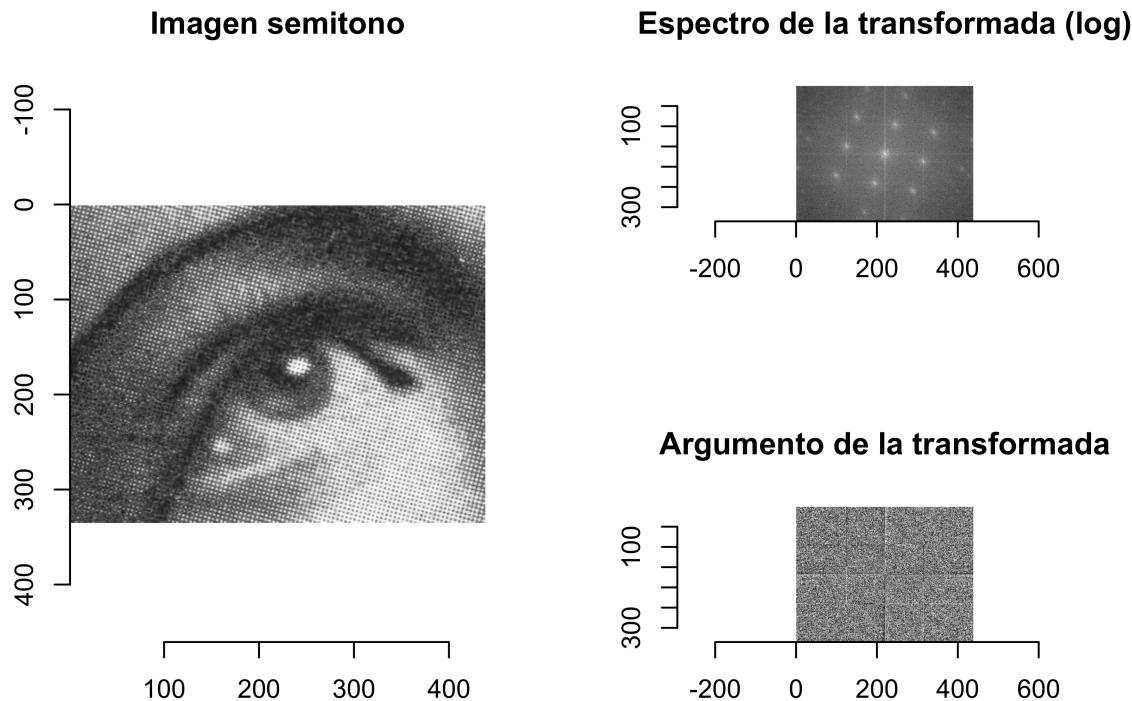


```
# Modulo
Mod.wfftC <- log(Mod(FFT.im))
# Fase
Fase.wfftC <- Arg(FFT.im)
```

Visualizamos y mostramos como quedan ahora las frecuencias en la transformada.

```
# cambiar los nombres de las variables

layout(matrix(c(1,1,2,3),ncol=2))
plot(HalfT,main="Imagen semitono")
plot(Mod.wfftC,main="Espectro de la transformada (log)")
plot(Fase.wfftC,main="Argumento de la transformada")
```



Se puede apreciar que al cambiar la imagen al dominio de frecuencias antes de aplicar la transformada estas aparecen ahora en el centro.

Diseño de la mascara para el filtrado

Primero umbralizamos el espectro de la transformada. Hay que elegir un valor de corte. Podemos probar diferentes valores y conocer los cuartiles nos puede ayudar a escoger.

Si queremos más opciones podemos mirar tambien los diferentes cuantiles.

Primer cuartil, 25% de los valores del conjunto están por debajo de este valor.
Dejaremos pasar el 25% de frecuencias.

Mediana, 50% de los valores del conjunto están por debajo de este valor.

Tercer cuartil, 75% de los valores del conjunto están por debajo de este valor.

Queremos quitar las frecuencias más altas que son menos abundantes por lo que dejaremos pasar el 85% de las frecuencias.

```
# algunas estadísticas para diseñar la máscara
fivenum(Mod.wfftC)
```

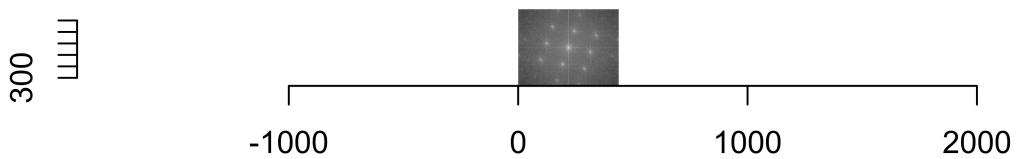
```
## [1] -2.447825 2.151771 2.756157 3.327368 10.648093
```

```
# UMBRAL

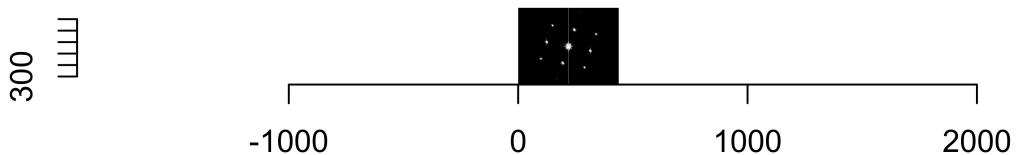
HalfT.FFT.Thresh <- imager::threshold(Mod.wfftC, thr = "99%")

layout(matrix(c(1,2),ncol=1))
plot(Mod.wfftC,main="Espectro de la transformada (log)")
plot(HalfT.FFT.Thresh,main="Espectro Binarizado")
```

Espectro de la transformada (log)



Espectro Binarizado



Diseñamos la mascara para quitar los valores de frecuencia que necesitamos

Despues de umbralizar vamos a retocar un poco el resultado para obtener la máscara.

Operaciones morfológicas a utilizar:

- Opening** hace desaparecer los objetos pequeños (generalmente blancos=1) y los convierte en fondo (generalmente negros=0)

- Closing** hace desaparecer los agujeros convirtiéndolos en parte del objeto

- Dilate** hace crecer el objeto (pixeles blancos)

```
# mmard

# Máscaras de diferentes tamaños y formas se pueden usar en las funciones morfológicas
# En este caso son todas cuadradas
kern1.square <- matrix(rep(1,9),ncol=3) # Tamaño 3x3
kern2.square <- matrix(rep(1,25),ncol=5) # Tamaño 5x5
kern3.square <- matrix(rep(1,49),ncol=7) # Tamaño 7x7
```

```

Open1 <- mmand::opening(HalfT.FFT.Thresh, kern1.square) # Escogemos kern1
Close2 <- mmand::closing(Open1,kern2.square) # Escogemos kern2

```

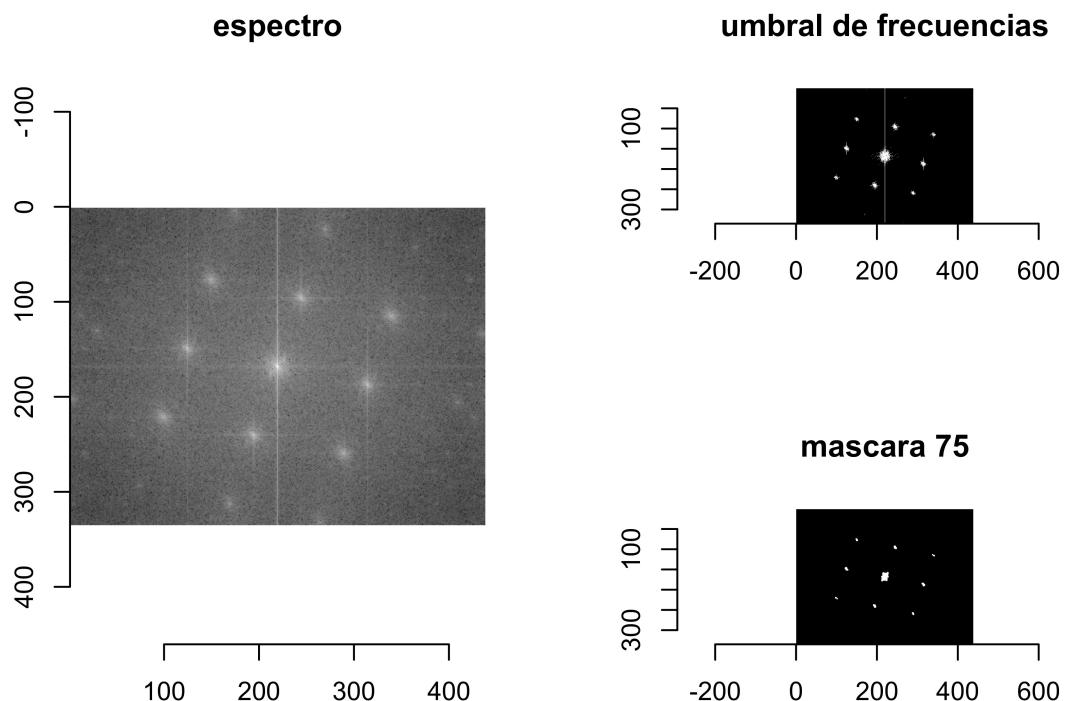
Vamos a mostrar ahora juntos el espectro, la imagen umbral y la máscara.

```

# imager
# cambiar los nombres de las variables

layout(matrix(c(1,1,2,3),ncol=2))
plot(Mod.wfftC,main="espectro")# No es tipo cimg original y fft stats!
plot(HalfT.FFT.Thresh,main="umbral de frecuencias")
plot(as.cimg(Close2),main=paste("mascara","75"))

```



Aplicación de la máscara

Recordamos que la máscara está compuesta de unos y ceros y que dependiendo de lo que queramos dejar pasar necesitamos unos o ceros en el lugar adecuado. En este caso vamos a tapar las frecuencias mas altas.

```

#
#mask.neg <- 1- Close2
mask.neg <- Close2

```

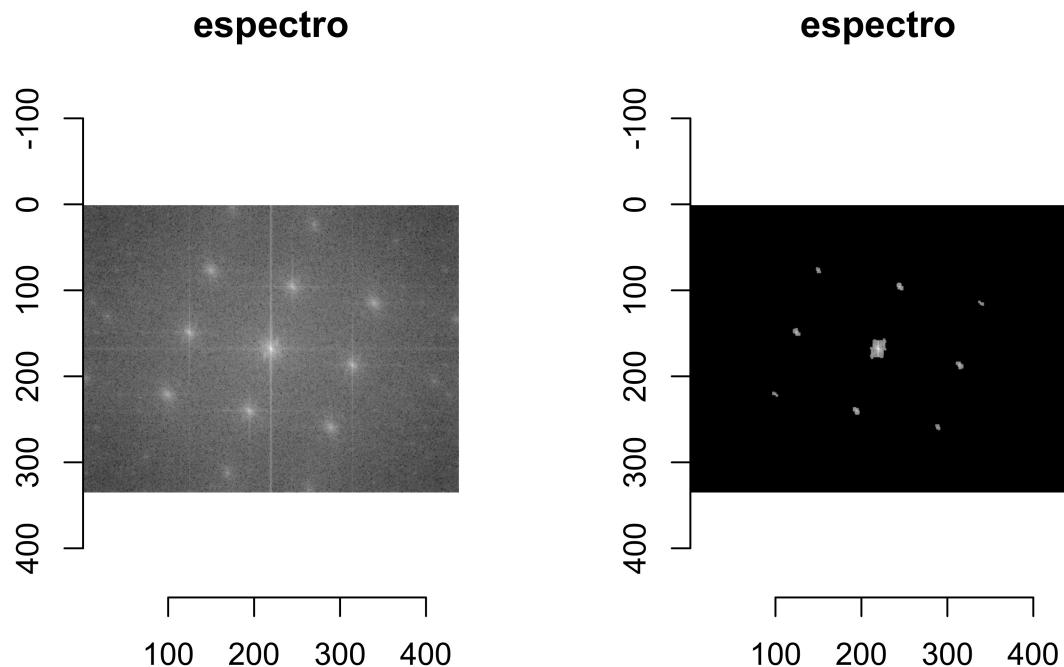
```

# dim(mask.neg)
# dim(Mod(FFT.im))
# dim(FFT.im)

Quita.freq <- mask.neg*Mod.wfftC
#Quita.freq <- mask.neg*HalfT.FFT.Thresh

layout(matrix(c(1,2),ncol=2))
plot(Mod.wfftC,main="espectro")# No es tipo cimg original y fft stats!
plot(Quita.freq,main="espectro")# No es tipo cimg original y fft stats!

```



Transformada inversa

Recuperamos la imagen de nuevo una vez quitadas las frecuencias en la transformada original, no en el espectro.

```

library("imager")
library("fftw")

## Warning: package 'fftw' was built under R version 4.1.2

##
## Attaching package: 'fftw'

```

```

## The following object is masked from 'package:imager':
##
##      FFT

library("fftwtools")
library("mmand")

## Warning: package 'mmand' was built under R version 4.1.2

##
## Attaching package: 'mmand'

## The following objects are masked from 'package:imager':
##
##      dilate, display, erode, threshold

# wfft.C, imagen fft
# dim(mask.neg)
# dim(Mod(FFT.im))
# dim(FFT.im)

matrix1.mask <- matrix(mask.neg,ncol = dim(FFT.im)[2],byrow = FALSE)

# Aplicar el filtro en el dominio de frecuencias
Quita.freq.img <- matrix1.mask*as.matrix(FFT.im)

# Recuperar la imagen
pru <- fftw(Quita.freq.img,inverse = 1,HermConj = 0,n=NULL )
matrix.pru <- matrix(pru,ncol = dim(FFT.im)[2],byrow = TRUE)
# dim(matrix.pru)
# dim(FFT.im)

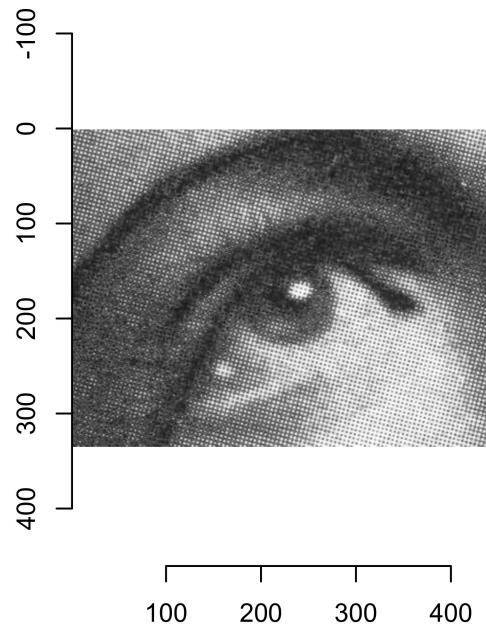
# Recuperar.im <- FFT(Re(Quita.freq.img,HermConj = 0,inverse = TRUE)
# Recuperar.im <- FFT(Re(Quita.freq.img),Im(Quita.freq.img),inverse = TRUE)

# cambiar los nombres de las variables

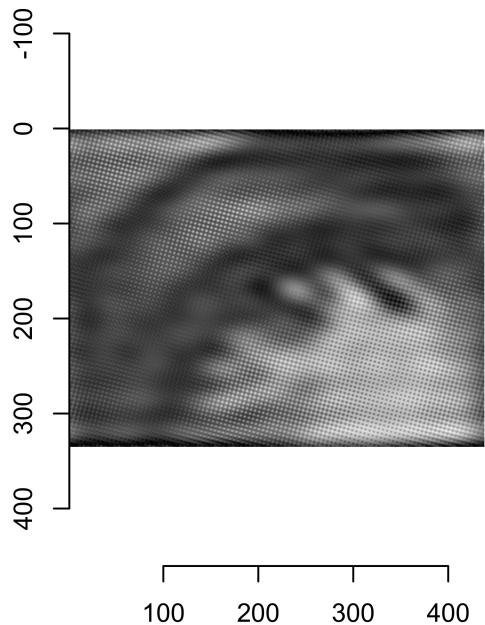
layout(matrix(c(1,1,2,2),ncol=2))
plot(HalfT,main="Imagen semitono")
#plot(Mod.imgHrecu,axes=FALSE,main="Después de aplicar el filtro")
plot(as.cimg(Mod(matrix.pru)),main="Despues de aplicar el filtro")

```

Imagen semitono



Despues de aplicar el filtro



Sólo parece aclarar un poco la imagen

Ejercicio 4

Cargamos los paquetes necesarios:

```
if(!require(seriation)){install.packages("seriation")}

## Loading required package: seriation

## Warning: package 'seriation' was built under R version 4.1.2

library(seriation)

library("EBImage")

## 
## Attaching package: 'EBImage'

## The following objects are masked from 'package:mmand':
## 
##     closing, dilate, display, erode, medianFilter, opening

## The following objects are masked from 'package:imager':
## 
##     channel, dilate, display, erode, resize, watershed

library("mmand")
```

Binarización de la imagen

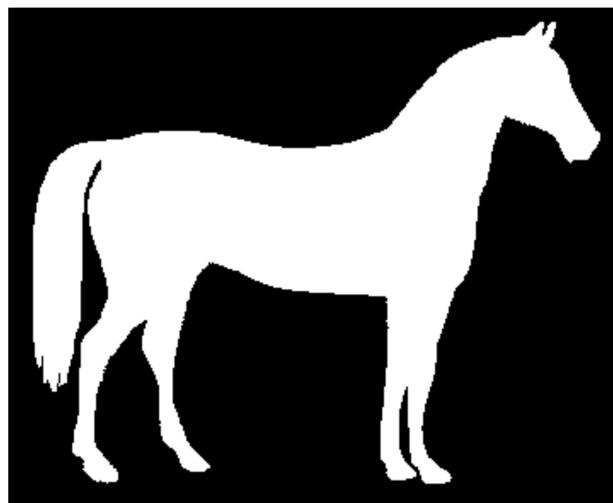
Cargamos la imagen de caballoB.jpg y pasamos a una matriz de unos y ceros (binaria) con dos bucles for.

```
img <- readImage("C:/Users/Javi/Desktop/pruebas/caballoB.jpg")
img <- channel(img, "grey")

myMat<-matrix(, ncol=356, nrow=292, byrow=TRUE)
for (i in 1:292){
  for (j in 1:356){
    if(imageData(img)[j,i]>=0.5){
      myMat[i,j]=1
    }
    else{
      myMat[i,j]=0
    }
  }
}
```

Imprimimos la matriz para corroborar que esta bien:

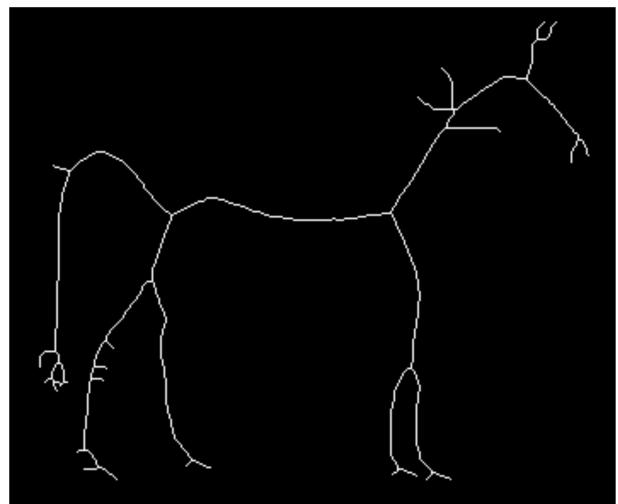
```
plot(as.raster(myMat))
```



Esqueletonizar

Ahora aplicamos el algoritmo de hit or miss. Para evitar implementarlo a mano ya que es iterativo y simple en funcionamiento utilizaremos la utilidad skeletonise del paquete “mmand”. Basicamente se aplica un operador morfológico iterativamente hasta transformar la imagen en esqueleto.

```
imagen<-skeletonise(myMat, kernel = NULL, method = "hitormiss")
plot(as.raster(imagen))
```



Conclusiones y dificultades

En éste trabajo hemos explorado diferentes técnicas de procesamiento de imagen desde la separación de una imagen en capas de bits hasta utilizar operaciones morfológicas para esqueletizar una imagen binaria. La principal dificultad encontrada ha sido realizar la esqueletización del caballo ya que implementar el algoritmo de hit and miss a mano era algo tedioso por lo que he optado por realizarla utilizando el paquete “mmand” y la función skeletonise.