Master: Ciencia de Datos e Ingeniería de Computadores

Curso: Visión por Computador

Rosa Mª Rodríguez Sánchez

Dpto. Ciencias de la Computación e I. A.

E.T.S. de Ingenierías Informática y de Telecomunicaciones

Universidad de Granada

Introducción al procesamiento de imágenes digitales con Matlab

Índice de contenido

1. Introducción

Este documento pretende resumir los aspectos que deben ser conocidos por los alumnos para realizar las prácticas de la asignatura con Matlab. Para conocer con mayor profundidad dichos puntos, se recomienda la consulta de los siguientes documentos:

- Javier García de Jalón, José Ignacio Rodríguez, Jesús Vidal. Aprenda Matlab 7.0 como si estuviera en primero. E.T.S. Ingenieros Industriales. Universidad Politécnica de Madrid. 2005 (http://www.tayuda.com).
- Manuales técnicos del propio Matlab. Descarga gratuita a través de la página web de Mathworks http://www.mathworks.com.
  - Índice de documentos: http://www.mathworks.com/access/helpdesk/help/helpdesk.html
  - Programación con Matlab: http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html
  - Toolbox de procesamiento de imágenes: http://www.mathworks.com/access/helpdesk/help/toolbox/images/

Se asume que el alumno ya conoce Matlab a nivel de programación así como técnicas de programación estructurada.

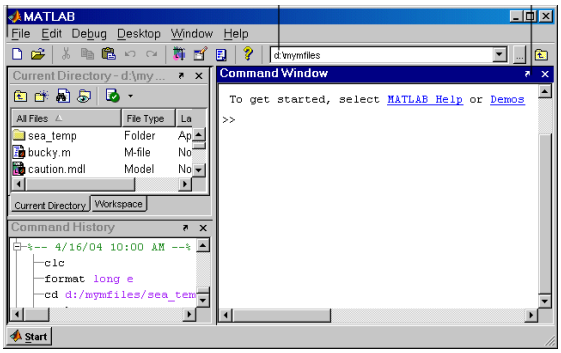
Además, en este documento también se inicia al alumno en el tratamiento de imágenes digitales con Matlab.

2. Sobre Matlab ...

2.1. ¿Qué es Matlab?

Matlab (acrónimo de MATrix LABoratory) es un entorno de programación orientado al cálculo matricial que dispone de una amplia colección de bibliotecas especializadas de funciones (toolboxes) para realizar tareas muy variadas (cálculo simbólico, estadística, procesamiento de señales, generación de gráficos, etc.).

En la figura de la derecha vemos la apariencia de este entorno al comienzo de su ejecución. Además de las barras de menú de la parte superior, disponemos de tres áreas claramente diferenciadas:



Command Window: En la parte derecha, la sub-ventana más amplia y en blanco. En esta zona es donde escribimos las instrucciones y visualizamos el resultado de su ejecución normalmente.

Command History: En la parte inferior izquierda. Tiene un historial con las instrucciones que hemos ido ejecutando durante la sesión de trabajo.

Current Directory / Workspace: En la parte superior izquierda. Muestra una lista de ficheros en el directorio de trabajo actual (Current Directory) o bien información sobre las variables que han sido definidas en la sesión de trabajo (Workspace).

2.1.1. Herramientas importantes

Help browser. Se accede a él en el menú “Help->MATLAB Help” (o pulsando F1). A través de esta herramienta tenemos acceso a todos los manuales on-line de matlab. Dispone de funciones que permiten realizar la búsqueda de palabras clave. Dentro de esta ventana podemos acceder a todas las demostraciones incluidas en el entorno que muestran, entre otras cosas, como se usan la mayor parte de las funcionalidades de los toolboxes.

También es posible obtener información sobre las instrucciones de Matlab usando la instrucción help seguida del nombre de la instrucción sobre la que deseamos ayuda. En lugar de help podemos usar la instrucción doc que abre navegador del sistema de ayuda (Help browser) mostrando la información solicitada.

Array editor. Se accede a él pulsando sobre el nombre de alguna variable de tipo matriz en el Workspace browser. Permite visualizar, de manera gráfica, los valores que almacena dicha matriz y modificarlos de una manera cómoda.

Editor / debugger. El depurador está disponible en el menú “Debug”. Permitirá realizar una ejecución paso a paso de los programas.

Profiler. Accesible en “Desktop->Profiler”. Es una herramienta que crea un informe sobre la ejecución de los programas destinado a mejorar su eficiencia.

2.2. Tipos de datos, operadores y expresiones

Los tipos de datos numéricos conocidos en Matlab son los siguientes:

- Enteros con signo: int8, int16, int32, int64
- Enteros sin signo (positivos): uint8, uint16, uint32, uint64
- Reales: simple, double

Por defecto, Matlab trata a todos los números como reales de doble precisión (double). Se utilizan 8 bytes para almacenar cada número (con 15 cifras decimales). Si queremos que un número sea tratado como uno de los otros tipos hemos de indicarlo expresamente.

Por ejemplo, para convertir un dato a alguno de esos tipos podemos usar la siguiente sintaxis:

```
x = int32(10);              % x es un entero de 32 bits
y = simple(x);              % y es un dato real simple
A = zeros(100,200);         % Matriz de 100x200 double
B = uint8(A);               % B es una matriz de 100x200 enteros de 8 bits
C = zeros(100,200,'int16'); % Matriz de 100x200 enteros de 16 bits
```

Para hacer consultas sobre el tipo de una variable (o de un dato) disponemos de las siguientes funciones:

- isinteger(dato). Devuelve true si el dato es entero (con o sin signo).
- isfloat(dato). Devuelve true si el dato es real (simple o double).
- isnumeric(dato). Devuelve true si el dato es entero o real.
- isa(dato,'tipo'). Devuelve true si el dato es del tipo indicado. Tipo será alguno de los tipos conocidos en Matlab (int8, uint16, double, logical, numeric, ...).
- class(dato). Devuelve el nombre del tipo de dato.

2.3. Matrices

Las matrices bidimensionales vienen dadas por filas. Para asignarles valores, estos se dan entre corchetes, separando cada fila con punto y coma. Por ejemplo:

A = [1 2 3 ; 4 5 6 ; 7 8 9]; % Fila 1: 1 2 3 Fila 2: 4 5 6 Fila 3: 7 8 9

Algunos de los operadores sobre matrices son los siguientes:

- A' calcula la traspuesta de A.
- A*B calcular la multiplicación matricial de A por B.
- inv(A) calcula la inversa de A.
- A+B sumar las matrices A y B.
- A – B resta las matrices A y B.
- A .* B multiplica, uno por uno, los elementos de A por los de B.
- A ./ B divide, uno por uno, los elementos de A entre los de B.

- A .\ B es igual que A ./ B.

El operador de división se usa para resolver sistemas de ecuaciones lineales. Por ejemplo, dado el sistema de ecuaciones Ax=B, la resolución del mismo viene dada por:

x = inv(A)*B

o bien:

x = A\B

Si el sistema de ecuaciones viene planteado en la forma xA=B, entonces la solución vendrá dada por:

x = B*inv(A)

o bien:

x = B/A

(observa que la barra de división es contraria al ejemplo anterior).

Además, es posible usar los operadores aritméticos habituales para operar matrices con escalares.

Algunas funciones interesantes relacionadas con matrices:

- eye(N). Crea una matriz identidad de NxN elementos.
- eye(N,M). Crea una matriz identidad de NxM elementos.
- ones(N). Crea una matriz de unos de tamaño NxN.
- ones(N,M). Crea una matriz de unos de tamaño NxM.
- zeros(N). Crea una matriz de ceros de tamaño NxN.
- zeros(N,M). Crea una matriz de ceros de tamaño NxM.
- rand(N). Crea una matriz de números aleatorios uniformemente distribuidos de tamaño NxN.
- rand(N,M). Crea una matriz de números aleatorios uniformemente distribuidos de tamaño NxM.
- size(A). Devuelve un vector con el tamaño de cada una de las dimensiones de la matriz.
- reshape(A,N,M). Devuelve una matriz de tamaño NxM tomando los elementos de A en orden (por filas).

2.3.1. Secciones de matrices

Mediante el operador de definición de rangos (:) es posible acceder a algunos trozos de una matriz (secciones). Este operador tiene la forma:

ini:fin

o bien,

ini:incr:fin

donde ini indica la posición inicial, fin la posición final e incr el incremento del índice.

Por ejemplo:

- x = 1:10 Asigna a x los elementos del 1 al 10, es decir, x vale [1 2 3 4 5 6 7 8 9 10]
- x = 1:2:10 Asigna a x los elementos [1 3 5 7 9]
- x = 10:-1.5:1 Asigna a x los elementos [10 8.5 7 5.5 4 2.5 1]

De esta forma, es posible seleccionar una sección de una matriz utilizando rangos en lugar de índices simples:

- A(1:5,7:-2:1) Toma las 5 primeras filas de la matriz, y de ellas, las columnas 7, 5, 3, 1

2.4. Programación en Matlab

2.4.1. Entrada/salida de datos interactiva

Para mostrar mensajes en consola usaremos la instrucción **disp**. Esta función lleva como argumento una matriz o una cadena de caracteres y la mostrará en consola.

```
disp('mensaje de texto')
disp(matriz)
```

Para pedir datos al usuario desde consola usaremos la función **input**. Esta función tiene como parámetro una cadena de caracteres con un mensaje para mostrar al usuario antes de la petición de datos. Devuelve el valor leído.

```
x = input('mensaje de texto')
```

Cualquier expresión válida en Matlab es aceptada como entrada y es evaluada antes de devolver el valor. Si no deseamos que se evalúe la entrada que damos entonces usaremos el parámetro adicional `'s'`:

```
x = input('mensaje de texto','s')
```

2.4.2. Estructuras de control

Las estructuras de control son las habituales en cualquier lenguaje de programación de alto nivel.

2.4.2.1. IF - Estructura condicional simple

```
if condición
      Instrucciones
end
```

2.4.2.2. IF-ELSE – Estructura condicional múltiple

```
if condición1
      Instrucciones1
elseif condición2
      Instrucciones2
elseif condición3
      Instrucciones3
...
else
      InstruccionesN
end
```

2.4.2.3. SWITCH – Estructura condicional múltiple

```
switch expresión
      case expr1,
            Instrucciones1
      case {expr2, expr3, ...}
            Instrucciones2
      ...
      otherwise,
            InstruccionesN
end
```

2.4.2.4. FOR - Estructura iterativa

```
for contador=ini:fin
      Instrucciones
end
```

```
for contador=vector_de_valores
      Instrucciones
end
for contador=ini:incr:fin
      Instrucciones
end
```

2.4.2.5. WHILE – Estructura iterativa

```
while condición
      Instrucciones
end
```

2.4.3. Los ficheros .m

Los ficheros con extensión .m son ficheros de texto ASCII que permiten guardar código escrito en Matlab. Hay dos tipos de ficheros:

- **Ficheros de tipo script**. Guardan secuencias de instrucciones de Matlab, tal y como las escribimos en la ventana de comandos. Al escribir su nombre (sin .m) en la ventana de comandos de Matlab, se ejecutan las instrucciones contenidas en el fichero en orden secuencial. Para que esto se pueda hacer, el fichero debe estar guardado en la carpeta de trabajo actual o bien en alguna de las carpetas contenidas en la ruta de búsqueda de Matlab. Las variables que se utilizan en el fichero pertenecen al espacio de trabajo actual del Matlab.
- **Ficheros de funciones**. Guardan funciones de Matlab. Cada fichero almacena una única función.

2.4.4. Funciones

Las funciones de Matlab de almacenan en ficheros con extensión .m. Cada función se almacena en un único fichero.

La primera línea de dicho fichero define el prototipo de la función con la siguiente sintaxis:

```
function [lista de valores devueltos] = nombre_función(lista de parámetros)
```

donde nombre_función es el nombre de la función que, además, debe coincidir con el nombre del fichero .m en el que se elmacena.

- Tanto la lista de parámetros como la lista de valores devueltos son listas de nombres de variables separadas por comas. Los corchetes son opcionales si sólo se devuelve un dato. También son opcionales los paréntesis si no hay parámetros.
- A efectos prácticos, todos los parámetros se consideran pasados por valor, esto es, una función no puede alterar el valor de los parámetros actuales de la llamada a la función.
- Todas las variables definidas dentro de la función son locales a la misma, es decir, su ámbito se restringe a dicha función y no son accesibles desde fuera de ella.
- Una función finaliza bien cuando se llega a su última instrucción ejecutable o bien cuando se ejecuta una instrucción `return`.

El número de parámetros y valores devueltos de una función puede ser variable. Para ello sustituiremos la lista de valores devueltos por `varargout` y/o la lista de parámetros de entrada por `varargin` en función de lo que deseemos que sea variable.

```
function varargout = nombre_funcion(varargin)
```

Para saber, desde dentro de la función, cuántos parámetros o valores devueltos tenemos usaremos, respectivamente, las variables `nargin` y `nargout`. Por ejemplo:

```
if nargin==2
      disp('Hay dos parámetros de entrada');
end
if nargout~=2
```

```
            disp('El número de valores devueltos es distinto de dos');
      end
```

Tanto varargin como varargout son vectores de celdas (cell array) por lo que el acceso a sus elementos se hace mediante el uso de llaves en lugar de paréntesis:

```
      if nargin>=1
            x = varargin{1};
      end
      if nargout==3
            varargout{1} = 3.1415;
            varargout{2} = 2.78;
            varargout{3} = 4.356;
      end
```

2.5. Otros detalles

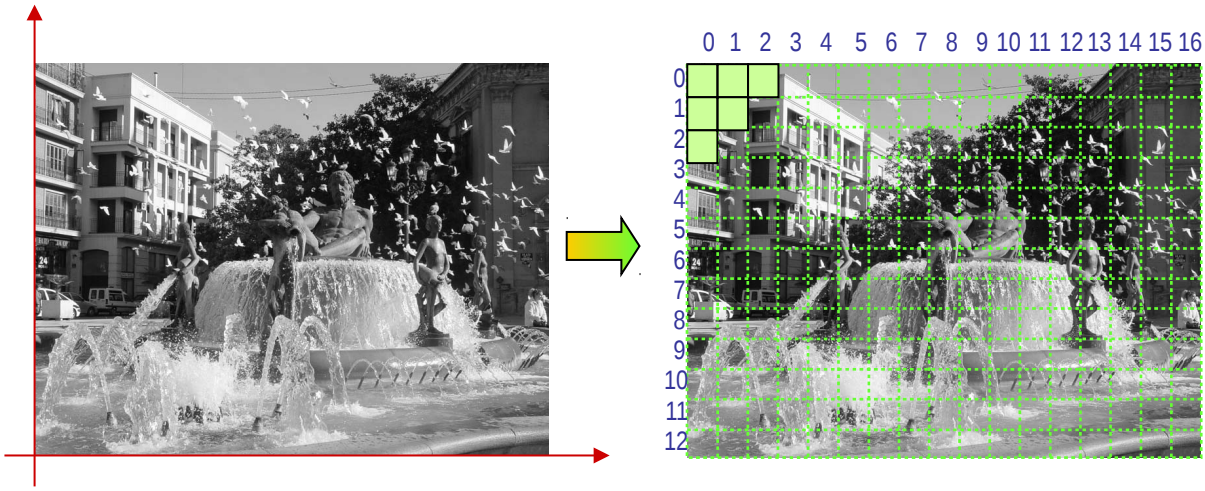
- Matlab distingue entre mayúsculas y minúsculas.
- Cuando una línea es demasiado larga y queremos continuar escribiendo en la línea siguiente podemos escribir ...
- Si escribimos ; al final de una instrucción, no se muestra el resultado de evaluar dicha instrucción en consola.
- Los comentarios van precedidos de %.
- La función clear permite borrar el espacio de trabajo actual. Se puede usar para borrar sólo algunas variables (y no todas).
- Las instrucciones save y load permiten guardar el espacio de trabajo actual y recuperarlo.

3. Imágenes en Matlab

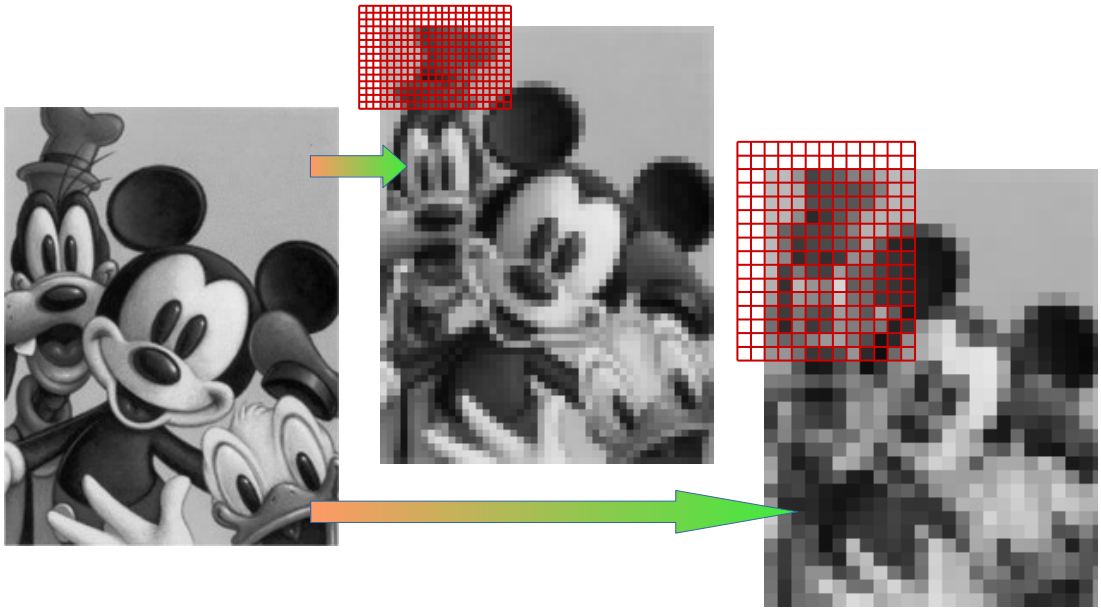
3.1. El proceso de digitalización de una imagen

Una imagen en el mundo real se puede ver como una función continua f(x,y). Tanto el dominio de x e y como los valores de f son continuos y reales. La digitalización de una imagen consiste en su transformación a una representación discreta. Esta discretización ha de hacerse tanto a x e y como a los valores f(x,y).

- El **muestreo** (*sampling*). La discretización de x e y se conoce como muestreo. El proceso de muestreo de una imagen consiste en representar sólo algunos de los valores x e y. De esta forma, tras muestrear una imagen del mundo real, obtendremos una matriz con un número finito de puntos (en x e y).



Dependiendo del número de puntos que tomemos al muestrear, la calidad de la imagen digital será mayor o menor. A continuación vemos un ejemplo en el que muestreamos una imagen con 2 patrones distintos, uno de ellos toma intervalos el doble de grandes que el otro.

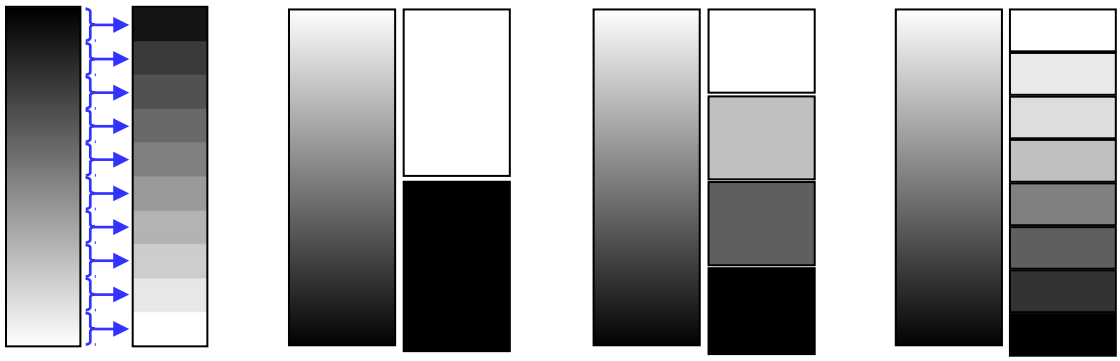


Cada celda de la matriz cubre un área de la imagen continua. El valor de esa celda debe representar a todos los puntos de la imagen continua que están en ese área. Podría ser uno de ellos al azar, el valor central, la media de todos ellos, etc..

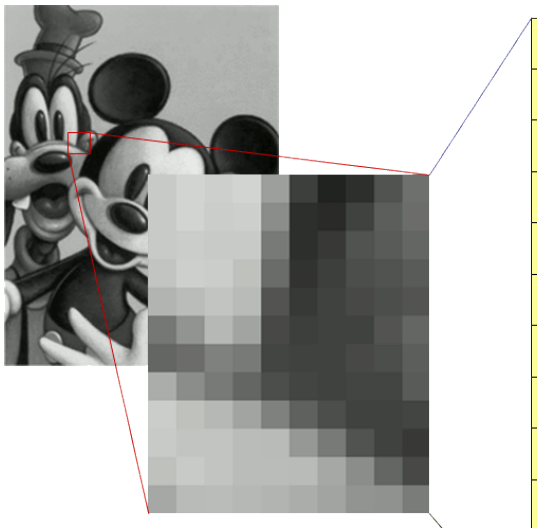
- La **cuantificación** (*quantization*). Tras muestrear la imagen, cada valor f(x,y) sigue siendo real y continuo. El proceso de cuantificación consiste en discretizar dichos valores.

  En la primera figura de abajo vemos un ejemplo en el que pasamos de un conjunto infinito de intensidades (cada intensidad se correspondería con un valor real) a un conjunto finito de intensidades (en concreto 10). Cada intensidad del conjunto discretizado representa un intervalo de intensidades en el conjunto original.

  En este proceso se pierde la posibilidad de representar algunos tonos pero es necesario para digitalizar la imagen. Dependiendo del número de niveles que elijamos para discretizar, tendremos una imagen digital de más o menos calidad. Abajo vemos ejemplos de cuantificación en 2, 4 y 8 niveles de un espacio continuo.



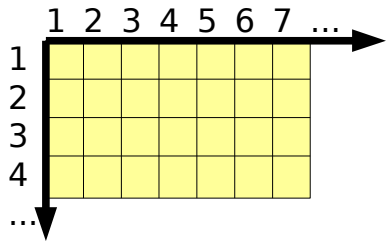
Tras el proceso de muestreo y cuantificación obtendremos la imagen digital, que no es más que una matriz de valores discretos. Cada uno de esos valores representa un valor de luminosidad. En la imagen que ves a continuación, vemos el detalle de una imagen digital en la que el intervalo de cuantificación es [0,255]. El color negro se representa con 0 y el color blanco con 255. Los valores intermedios representan, de manera proporcional, a los tonos intermedios. A cada uno de estos valores se le denomina **píxel** (del inglés *Picture Element*).



202	211	208	211	158	62	34	47	82	112
202	211	204	208	140	47	42	66	93	109
202	204	202	202	121	47	57	82	90	101
195	208	204	192	107	51	62	77	82	90
179	188	195	186	84	55	66	73	77	84
119	148	183	163	73	62	66	66	84	101
101	109	128	121	68	66	66	73	77	93
173	140	121	101	77	68	66	68	68	90
204	192	183	163	128	96	77	66	66	68
202	195	195	188	186	152	121	82	66	57
192	202	195	188	188	186	167	140	101	73
167	186	188	183	179	173	163	148	146	123

3.2. ¿Qué es una imagen para Matlab?

En Matlab, una imagen digital no es más que una matriz de números reales. En esa matriz, el primer pixel de la imagen se halla en las coordenadas (1,1), que se corresponde con la esquina superior izquierda de dicha matriz tal como vemos en la figura de la derecha. En otros lenguajes de programación, los índices de las matrices comienzan su numeración en cero.



Matlab puede trabajar con varios tipos de imágenes:

- De intensidades (niveles de gris). Cada pixel contiene un valor dentro de un determinado rango que representa una intensidad luminosa (un nivel de gris entre el negro y el blanco). El rango concreto de valores varía dependiendo del tipo de dato del que esté formada la matriz. Por ejemplo, si la matriz contiene datos de tipo uint8, entonces el rango de valores será [0,255]. Si la matriz es de tipo uint16, entonces el rango de valores será [0,65535]. Aunque, tal como hemos visto, los valores cuantificados son enteros, Matlab está pensado para trabajar con números reales (double) y es por eso que permite tratar matrices de reales como si fuesen imágenes. En ese caso, el intervalo de valores que se considera es siempre [0,1].
- Binarias. Cada pixel contiene un valor de entre dos posibles. El tipo de la matriz debe ser logical.
- De color (RGB). Son imágenes en color representadas por matrices 3D.
- Indexadas. Se usan para representar imágenes en color haciendo uso de una paleta o mapa de color.

3.2.1. ¿Cómo se almacena una imagen en disco?

Aunque desde un punto de vista práctico una imagen es una matriz, a la hora de almacenar esos datos en disco, el tema se complica un poco. Lo normal es que esos datos sufran distintos tipos de transformaciones antes de ser guardados en un fichero. La razón suele ser siempre la misma: intentar que la imagen ocupe el menor espacio posible. En este sentido podemos distinguir entre dos tipos de formatos:

- Sin pérdidas. Los que permiten almacenar la imagen de manera exacta (con compresión o sin compresión). Por ejemplo PNG (Portable Network Graphics), PGM (Portable Gray Map), PPM (Portable Pixmap Format), BMP (Windows Bitmap), GIF (Graphics Interchange Format).
- Con pérdidas. Los que, a cambio de reducir más el espacio que ocupa en disco la imagen, pierden parte de los datos que hay en dicha imagen. Por ejemplo JPG (Joint Photographic Experts Group).

Hay algunos formatos que admiten distintos algoritmos para guardar la imagen con o sin pérdidas en función de lo que decida el usuario. Por ejemplo TIFF (Tagged Image File Format).

Matlab permite trabajar,entre otros, con los formatos: BMP, GIF, JPG, PBM, PGM, PNG, PNM, PPM, y TIFF. Dependiendo de la versión de Matlab esto puede variar por lo que podemos averiguar esta información usando la instrucción imformats:

```
>> imformats
EXT       ISA     INFO        READ      WRITE     ALPHA  DESCRIPTION
-----------------------------------------------------------------------------------------------
bmp       isbmp   imbmpinfo   readbmp   writebmp  0      Windows Bitmap (BMP)
cur       iscur   imcurinfo   readcur             1      Windows Cursor resources (CUR)
fts fits  isfits  imfitsinfo  readfits            0      Flexible Image Transport System (FITS)
gif       isgif   imgifinfo   readgif   writegif  0      Graphics Interchange Format (GIF)
hdf       ishdf   imhdfinfo   readhdf   writehdf  0      Hierarchical Data Format (HDF)
ico       isico   imicoinfo   readico             1      Windows Icon resources (ICO)
jpg jpeg  isjpg   imjpginfo   readjpg   writejpg  0      Joint Photographic Experts Group (JPEG)
pbm       ispbm   impnminfo   readpnm   writepnm  0      Portable Bitmap (PBM)
pcx       ispcx   impcxinfo   readpcx   writepcx  0      Windows Paintbrush (PCX)
pgm       ispgm   impnminfo   readpnm   writepnm  0      Portable Graymap (PGM)
png       ispng   impnginfo   readpng   writepng  1      Portable Network Graphics (PNG)
pnm       ispnm   impnminfo   readpnm   writepnm  0      Portable Any Map (PNM)
```

```
ppm       isppm   impnminfo   readpnm   writepnm  0      Portable Pixmap (PPM)
ras       isras   imrasinfo   readras   writeras  1      Sun Raster (RAS)
tif tiff  istif   imtifinfo   readtif   writetif  0      Tagged Image File Format (TIFF)
xwd       isxwd   imxwdinfo   readxwd   writexwd  0      X Window Dump (XWD)
```

Si deseamos saber las características de una imagen concreta almacenada en un fichero podemos usar la instrucción imfinfo. Por ejemplo, supongamos que tenemos una imagen almacenada en un fichero llamado mickey-goofy-donald.gif:

```
>> imfinfo('mickey-goofy-donald.gif')
ans =
            Filename: 'mickey-goofy-donald.gif'
         FileModDate: '21-nov-2006 09:45:05'
            FileSize: 75784
              Format: 'GIF'
       FormatVersion: '89a'
               Width: 244
              Height: 320
            BitDepth: 8
           ColorType: 'indexed'
     FormatSignature: 'GIF89a'
     BackgroundColor: 0
         AspectRatio: 0
          ColorTable: [256x3 double]
          Interlaced: 'no'
    CommentExtension: 'Created with The GIMP '
```

Esta instrucción devuelve una estructura que contiene campos con el nombre del fichero, el formato de almacenamiento, el tamaño de la imagen, la profundidad de color, etc.. La información devuelta dependerá del formato de la imagen. En el ejemplo anterior la imagen era de tipo GIF, pero si, por ejemplo, preguntamos sobre una imagen que está almacenada con el formato TIFF, el resultado sería el siguiente:

```
>> imfinfo('mickey-goofy-donald.tif')
ans =
                    Filename: 'mickey-goofy-donald.tif'
                 FileModDate: '21-nov-2006 11:43:02'
                    FileSize: 72806
                      Format: 'tif'
               FormatVersion: []
                       Width: 244
                      Height: 320
                    BitDepth: 16
                   ColorType: 'grayscale'
             FormatSignature: [73 73 42 0]
                   ByteOrder: 'little-endian'
              NewSubfileType: 0
               BitsPerSample: [8 8]
                 Compression: 'LZW'
    PhotometricInterpretation: 'BlackIsZero'
                StripOffsets: [20x1 double]
             SamplesPerPixel: 2
                RowsPerStrip: 16
             StripByteCounts: [20x1 double]
                 XResolution: 72
                 YResolution: 72
              ResolutionUnit: 'None'
                    Colormap: []
          PlanarConfiguration: 'Chunky'
                   TileWidth: []
                  TileLength: []
                 TileOffsets: []
              TileByteCounts: []
```

```
        Orientation: 1
          FillOrder: 1
    GrayResponseUnit: 0.0100
      MaxSampleValue: [255 255]
      MinSampleValue: 0
        Thresholding: 1
    ImageDescription: 'Created with The GIMP'
            Software: [1x58 char]
        ExtraSamples: 2
```

3.2.2. Leyendo, visualizando y guardando imágenes

Para leer una imagen se usa la instrucción imread. La sintaxis más habitual es la siguiente:

```
A = imread('imagen.ext');
```

donde imagen.ext es el nombre del fichero en disco. El resultado de la función es una matriz. Si la imagen es de intensidades (niveles de gris), la matriz es bidimensional pero si la imagen es en color la matriz será tridimensional. La clase de la matriz dependerá de los datos almacenados en el fichero (podría ser uint8, uint16 o logical si la imagen es binaria).

Por ejemplo, podemos cargar la imagen mickey-goofy-donald.png y almacenarla en la matriz A:

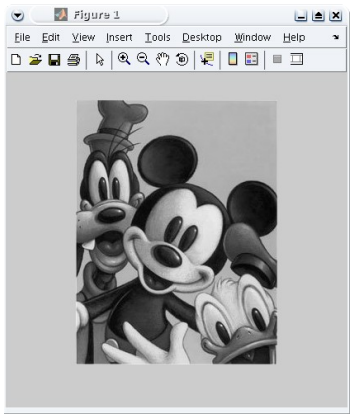
```
>> A = imread('mickey-goofy-donald.png');
>> whos A
  Name      Size                    Bytes  Class
  A       320x244                   78080  uint8 array
  Grand total is 78080 elements using 78080 bytes
```

La instrucción whos permite conocer información sobre el tamaño de la matriz y de que clase son sus elementos. En este caso vemos que la matriz es 2D y tiene 320 filas y 244 columnas. Cada elemento de la matriz es de tipo uint8.

Si omitimos el punto y coma al ejecutar imread, veremos en la consola todos los valores que devuelve la función por lo que se suele poner siempre para evitarlo.

Una vez cargada la imagen, podemos visualizarla con la instrucción imshow:

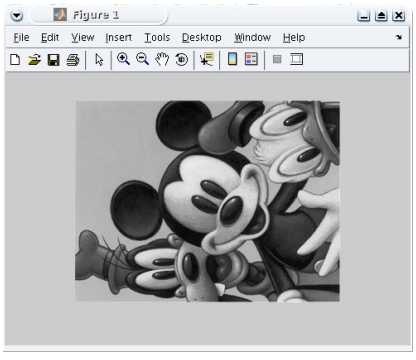
```
>> imshow(A)
```



Como resultado se abre una nueva ventana en la que se muestra la imagen tal como vemos en la figura de la derecha.

Tras cargar la imagen, además de visualizarla, podemos realizar transformaciones sobre ella. Por ejemplo podemos rotarla 90 grados hacia la izquierda con la instrucción imrotate obteniendo una nueva imagen (más adelante estudiaremos con más detalle esta y otras funciones para hacer transformaciones geométricas sobre imágenes):

```
>> b=imrotate(a,90);
>> imshow(b);
```



En la figura de la derecha vemos la nueva imagen tras la rotación.

En este ejemplo nos habremos dado cuenta de que al ejecutar la instrucción imshow(b), se ha borrado de la ventana la primera imagen y a continuación se ha dibujado la segunda imagen. Si queremos visualizar ambas imágenes simultáneamente usaremos la instrucción figure. Esta instrucción crea una nueva ventana para dibujar gráficos dejando abiertas las que hubiese hasta ese momento. Desde el momento que se ejecuta figure, cualquier nueva visualización se hará sobre la nueva ventana.

```
>> imshow(a);
>> figure
>> imshow(b);
```

Vemos que cada vez que abrimos una nueva ventana de dibujo, se numera de forma consecutiva. Podemos usar la instrucción figure para cambiar la ventana activa en lugar de abrir nuevas ventanas. Por ejemplo, si ejecutamos:

```
>> figure(1)
>> imshow(b);
```

lo que conseguimos es, primero activar la ventana número 1 (la que tiene la imagen a) y a continuación dibujar en esa ventana la imagen b.

Además de imshow, también podemos usar imtool e imagesc para visualizar imágenes.

Una vez que hemos acabado de manipular la imagen, podemos guardarla en en un fichero de disco, con la instrucción imwrite indicando, simplemente, el nombre del fichero. Según la extensión que utilicemos, se guardará en uno u otro formato:

```
>> imwrite(b,'rotada.png')
```

Disponemos de otros visualizadores:

- imagesc(img). Hace un escalado del rango dinámico de la imagen img y la visualiza.
- imagesc(img,[inf sup]). Permite indicar dos límites para visualizar sólo parte del rango dinámico.
- imtool. Visualizador que incluye varias herramientas (zoom, contraste, ...).

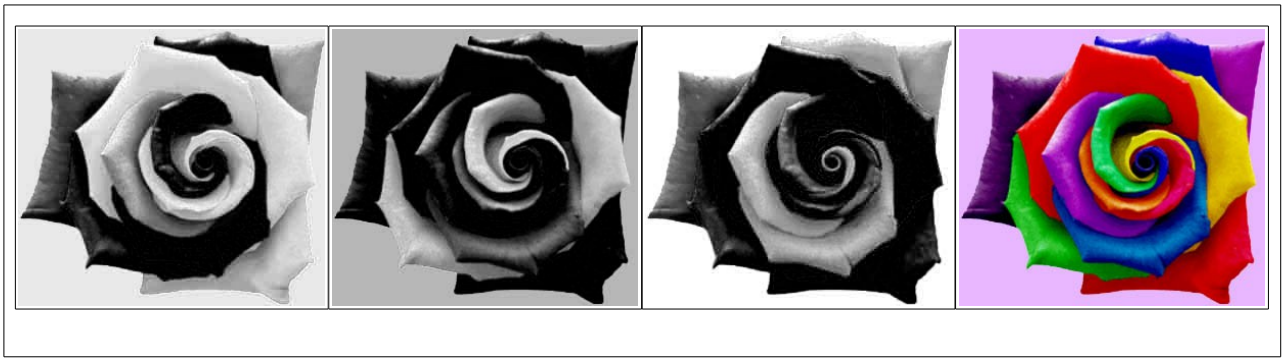
3.3. Las imágenes en color

Matlab permite trabajar con dos tipos de imágenes en color: imágenes RGB e imágenes indexadas. Las imágenes RGB se representan con matrices tridimensionales de M x N x 3. Cada pixel se representa por una tripleta de valores RGB indexados en la tercera dimensión de la matriz. Los valores que se almacenan en la matriz pueden ser, al igual que ocurre con las imágenes de niveles de gris, de tipo double, uint8, uint16, etc..

Esta matriz 3D se puede ver como 3 matrices 2D, cada una de ellas conteniendo los valores de una de las bandas R, G o B. En la terminología de Matlab, a cada una de estas bandas se la conoce como imagen componente (component image).

Durante el procesamiento de imágenes en color es frecuente tratar por separado las distintas bandas de color. Por lo tanto, es necesario, dada una imagen en color, obtener por separado cada una de las bandas y viceversa: dadas cada una de las bandas como una matriz 2D, componerlas para formar una única imagen RGB. Para formar una imagen RGB a partir de tres imágenes de niveles de gris se usa la función cat. Por ejemplo, dadas las imágenes de niveles de gris imR, imG e imB, podemos obtener una nueva imagen imRGB ejecutando:

```
>> imRGB = cat(3, imR, imG, imB);
```



La instrucción cat no es específica para trabajar con imágenes. Esta función se usa, en general, para concatenar matrices a lo largo de alguna dimensión.

Una vez que tenemos la imagen RGB podemos volver a descomponerla en sus tres componentes individuales usando los operadores de extracción de secciones de matrices:

```
>> imR = imRGB(:,:,1);
>> imG = imRGB(:,:,2);
>> imB = imRGB(:,:,3);
```

3.4. Conversión entre distintos tipos de imágenes

Matlab proporciona funciones para convertir la clase de las imágenes de unos tipos a otros. La sintaxis es:

```
>> imgResult = nuevo_tipo(imgOrig)
```

Donde nuevo_tipo es la nueva clase a la que queremos convertir la imagen imgOrig. Por ejemplo:

```
>> im1 = double(imorig)
>> im2 = uint8(im1)
```

Si la nueva clase tiene un rango de valores más limitado, entonces Matlab satura al mínimo o al máximo todos aquellos valores que estén fuera del rango válido para la nueva clase. Por ejemplo, si convertimos de double a uint8, cualquier valor superior a 255 se almacenará como 255 y cualquier valor inferior a cero se convertirá a cero. Si queremos que se realice un escalado de los valores antes de la conversión para evitar la perdida de datos que podrían ser relevantes, podemos usar algunas funciones específicas tales como im2uint8, im2uint16, im2double, im2bw y mat2gray.

La función rgb2gray convierte una imagen en color a niveles de gris quedándose únicamente con la componente de intensidad del modelo YIQ.

4. Ejercicios

1. Probar los distintos visualizadores de que disponemos en Matlab (imshow, imtool, imagesc). Para ello cargar la imagen disney.png y:
    1. Visualizarla con los tres visualizadores.
    2. Convertir a tipo double con double(img) y visualizar de nuevo.
    3. Convertir a double con im2double y analizar el resultado.
2. Descomponer una imagen RGB en sus tres componentes de color (rosa.jpg).
    1. Visualizar las tres componentes de manera simultánea junto con la imagen original y analizar los resultados.
    2. Anula una de sus bandas (por ejemplo la roja) y analiza los resultados. Se recomienda usar imtool para ver los valores de color en cada píxel.
    3. Usa también la imagen sintetica.jpg, haz otras modificaciones y observa los resultados (por ejemplo: poner una de sus bandas al nivel máximo, intercambiar el papel de las bandas entre sí, aplicar un desplazamiento a alguna de las bandas con circshift, invertir alguna de sus bandas con fliplr o flipud, ...).