Sistemas de Visão e Percepção Industrial

Aula Prática nº 1

Considerações gerais. Revisões de Matlab.
Exercícios básicos de relação entre imagens e matrizes.

Universidade de Aveiro
Vitor Santos, 2022-2023

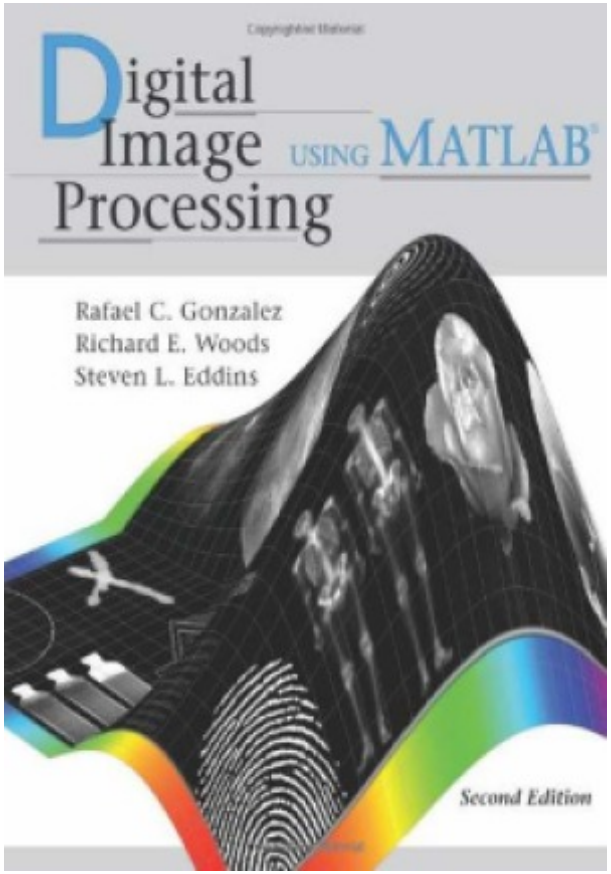
Sumário

Paradigma geral das aulas práticas

- Componente prática orientada para a programação!
  - Tema principal: A imagem – grande riqueza e complexidade.
  - Matlab (Versão oficial da UA - v2019b ou posterior)
    - Ferramentas genéricas
    - Toolbox de Processamento de Imagem
    - Toolbox de Aquisição de Imagem
    - Toolbox de Visão por Computador
    - Existem outras toolboxes relacionadas com imagem e visão artificial mas não deverão ser necessárias.
  - Software de Visão Industrial
    - Sherlock
- As competências na componente prática obtém-se fazendo muitos programas e exercícios.

Livro de Matlab para processamento de imagem

- Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins - Digital Image Processing Using Matlab. Gatesmark Publishing, 2nd Ed, 2009.
- Não será seguido rigorosamente nas aulas, mas contém muita matéria, exemplos e exercícios adicionais que podem ser úteis no estudo.



- Há uma primeira edição de 2004 que também serve.
- Há ainda uma 3ª edição de 2020 que inclui capítulos sobre técnicas mais avançadas que não serão abordadas nas aulas.

Metodologia dos exercícios nas aulas

- Criar uma pasta por aula (aula1, aula2, etc.)
- Criar uma pasta **lib** a usar ao longo do semestre
  - Nessa pasta serão colocadas funções úteis para várias aulas distintas
- Há duas metodologias principais para resolver os exercícios na aula:
  - Cada exercício está num ficheiro diferente:
    - a1ex1.m , a1ex2.m, etc.
  - Vários exercícios no mesmo ficheiro:
    - aula1.m, etc.
  - Os dois métodos têm vantagens e desvantagens.
    - Recomenda-se e adota-se o **segundo** caso para as aulas de SVPI.

Realização dos exercícios em ficheiro único

- Na opção a usar (todos os exercícios num só ficheiro), os exercícios devem ficar em secções distintas, demarcadas com comentários duplos, como no seguinte exemplo com dois exercícios:

```
1
2 -   %% Ex1
3 -   Z=zeros(100,200);
4 -   imshow(Z)
5
6 -   %% Ex2
7 -   X=ones(100,200);
8 -   imshow(X)
```

- As secções podem ser executadas individualmente (CTRL+ENTER).
  - NB. Pode haver o risco de interferência de variáveis entre as diversas secções. Se não for para usar variáveis de umas secções para outras, recomenda-se um comando 'clear' no início da secção para limpar as variáveis anteriores.
  - Cada secção deve iniciar com o comentário apropriado a identificar o exercício, como por exemplo: `%% Ex1`, `%% Ex 3b`, etc. Sem isso, o exercício poderá não ser devidamente interpretado ou até avaliado!

Outra forma de realizar exercícios num só ficheiro

- Pode-se criar uma lista de exercícios e executá-los condicionalmente:

```
1   % Aula N
2   exlist = {  %uncomment the name(s) of exercise(s) to run
3   %   'ex1',
4   %   'ex2',
5       'ex3'
6       };
7   %=========================================
8   if ismember('ex1', exlist)
9   %% ex 1
10  close all;
11  A1=imread('rice.png'); imshow(A1)
12  end
13  if ismember('ex2', exlist)
14  %% ex 2
15  close all;
16  A2=zeros(100,100); imshow(A2)
17  end
18  if ismember('ex3', exlist)
19  %% ex 3
20  close all;
21  A3=ones(100,150); imshow(A3)
22  end
```

- No exemplo, só o 'ex3' executa na execução do script (**F5**).
- Esta forma pode ser compatível com a anterior (CTRL+ENTER) mas é mais adequada à deteção de erros de escrita de código.

Outros exercícios e ficheiros

- Em certas situações, além do ficheiro com os programas (script file) também será preciso criar funções auxiliares.
- Nesse caso, as funções deverão ficar cada uma num ficheiro separado, na pasta de trabalho da aula ou, em alguns casos, na pasta **lib** criada.
- Muito comandos mais simples podem também ser executados na linha de comandos da janela principal do Matlab, e será por aí que se começará.

Imagens e matrizes

- Em Matlab, uma imagem representa-se por uma matriz
- pixels da imagem $\Longleftrightarrow$ elementos da matriz
- A numeração de linhas/colunas começa em 1
- A(3,4) $\rightarrow$ Valor do elemento na 3ª linha a contar de cima, e 4ª coluna a contar da esquerda.
  - O índice 'end' representa o último elemento da linha/coluna da matriz. Portanto, se A tiver 10 linhas e 20 colunas, o seguinte é verdadeiro:
    - A(3,20) == A(3,end)
    - A(10,20) == A(end,end)

Tipos de imagem

- Há 4 tipos principais de imagens com significado variável, conforme o valor dos pixels e tipo de elementos da matriz:
  - Imagem a níveis de cinzento (ou de um só canal)
  - Imagem binária (preto/branco) (caso particular da anterior)
  - Cores (a estudar mais tarde) (com vários canais, em geral 3)
  - Indexada como o formato GIF (pouco importante para SVPI porque na verdade é convertida num dos outros formatos quando lida)
- Nesta fase, serão abordadas apenas as duas primeiras.
- Mais detalhes podem ser obtidos junto da Mathworks[1].

[1] `http://www.mathworks.com/help/images/image-types-in-the-toolbox.html`

Tipos de imagem principais e valores dos pixels

- O tipo e gama de valores dos pixels definem as tipologias de imagens
- As mais usadas na disciplina são as seguintes:
- Níveis de cinzento – os valores de um pixel p podem ser:
  - $p \in \{0,1,\ldots, 255\}$ se for do tipo **uint8**
  - $p \in [0,1]$ se for do tipo 'double'; $p \leq 0 \rightarrow$ preto, $p \geq 1 \rightarrow$ branco
    - Nota: também existe o tipo 'single' que tem menos casas decimais. Em geral não será muito útil em SVPI, sendo o 'double' o tipo por defeito para números decimais.
- Imagem binária (preto/branco)
  - $p \in \{0,1\}$ ou seja, do tipo 'logical'
- Imagem a cores (3 canais)
  - Os 3 canais serão de um dos tipos anteriores (**uint8** ou **double**)

Scripts e funções

- Os ficheiros principais a desenvolver e usar em Matlab são de dois tipos, e sempre com extensão '.m':
  - Scripts
    - Código de matlab que é essencialmente um conjunto de instruções que poderiam ser executadas na linha de comando.
    - As variáveis criadas ficam definidas com o seu último valor para todas as operações seguintes por linha de comando ou por outros scripts.
  - Funções (functions) - Ficheiros similares aos scripts, mas com as seguintes diferenças:
    - A primeira linha de código de uma *function* tem a keyword **function** cujo nome deve ser igual ao do ficheiro (sem extensão)
    - As funções podem devolver (retornar) valores que podem ser atribuidas a variáveis ou ser usadas por outras operações ou functions.
    - As funções podem aceitar parâmetros (argumentos) que serão usados nos seus cálculos internos.
    - Os argumentos das funções podem ter qualquer nome e a sua eventual modificação não altera as variáveis existentes antes da chamada da função
    - As variáveis internas extinguem-se quando a função termina (retorna).

Live Scripts

- Há um terceiro tipo de ficheiros que são os Live Scripts ('.mlx')
- Qualquer script ('.m') pode ser aberto em modo Live Script
- Algumas questões sobre os Live Scripts:
  - permite adicionar ao programa elementos gráficos para enriquecer ou ilustrar o documento (imagens, equações, hiperligações, etc.), por exemplo para criar relatórios (exportam em PDF, LaTeX, HTML, etc.)
  - representa os resultados da execução (gráficos, imagens, etc.) junto do código respetivo (por baixo ou ao lado).
  - pode ser guardado como um script normal, mas os *outputs* das execuções não figuram no ficheiro '.m', e perdem-se imagens.
  - é útil para se compreender o processo de execução do exercício (permite controlos interativos como botões, controlos deslizantes, listas, etc.)
  - é em geral mais lento na execução do que um script normal.
- Para entrega nas aulas práticas só são aceites scripts normais.
  - Se porventura algum trabalho for feito em Live Script terá de ser convertido em script normal antes da entrega.

Valores e variáveis

- Em Matlab os valores e variáveis são genericamente matrizes:
  - Se tiverem dimensão de 1x1 são escalares normais;
    - Números usuais.
  - As matrizes usuais de são duas dimensões (linhas e colunas).
    - Caso das imagens a níveis de cinzento
  - Mas as matrizes podem ter mais do que duas dimensões (Tensores ou 'N-D arrays').
    - Exemplo das imagens a cores que se representam em matrizes de 3 dimensões
- Em matlab as variáveis podem ter campos para ajudar a sua identificação/nomenclatura.
  - Por exemplo, pode-se criar uma variável 'a' com 2 campos:
    - a.img=zeros(100,100);
    - a.author='manuel';
- As variáveis com campos são conhecidas por **estruturas**

Construções de código em matlab

- As contruções em matlab são essencialmente de dois tipos:
  - Ciclos (**for-end** ou **while-end**)
  - Testes (**if-end** ou **switch-case-end**)
    - Os testes **if-end** ainda podem ter variantes **if-else-end** ou **if-elseif-end-end**
- Há keywords especiais que alteram a execução de ciclos (não de **if**s):
  - **break** aborta um ciclo prematuramente.
  - **continue** "salta" imediatamente para a próxima iteração do ciclo.
- Também é possivel programar em matlab de forma "object-oriented" que tem muitas vantagens, mas requer conceitos específicos!
  - Por isso deixa-se como mera sugestão aos mais interessados pela área da programação:
    - https://www.mathworks.com/help/matlab/object-oriented-design-with-matlab.html

Exercício inicial de Matlab – a realizar na linha de comandos da janela principal.

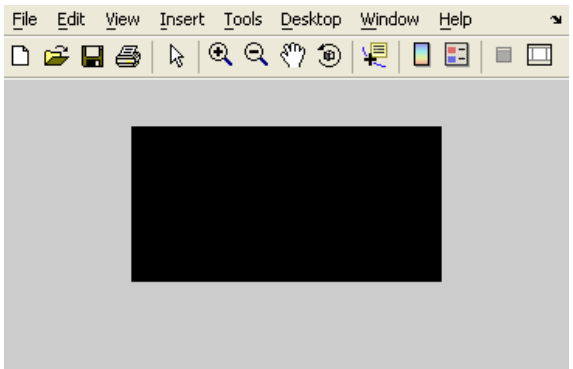
- Carregar uma imagem a níveis de cinzento para dentro de uma matriz (números de 0 a 255)
- Visualização da imagem que lhe corresponde
- Numa única operação separar os bagos de arroz do fundo da imagem escura:
  - Em vez de fazer um ciclo por cada pixel da imagem (linhas e colunas), pode-se fazer numa só operação e transformar uma imagem para dois níveis apenas (branco e preto):

```
A=imread('rice.png');
imshow(A)
B=255*(A>128);      %Comentar o procedimento
figure(2)           %para que serve?
imshow(B)           %Comentar o resultado.
```

Exercício 1a) - Criar uma imagem vazia

- Criar um script Matlab (por exemplo o ficheiro aula1.m) onde o código será escrito (usar o menu ou o comando edit na linha de comando para criar o ficheiro).
  - Os nomes dos ficheiros .m a criar **não** devem ter nomes com espaços, hífens (-) ou carateres acentuados, nem devem começar por algarismos!!!
- Escrever código para implementar a seguinte operação:
  - Sintetizar uma imagem 100×200 preta e vizualizá-a.
- Recordar a relação entre linhas e colunas e as dimensões da imagem
- Os comandos podem ser escritos diretamente na linha de comando um por um ou, preferencialmente, escritos num ficheiro para permitir a fácil repetição da execução.

```
%% Ex1a
Z=zeros(100,200);
imshow(Z)
```

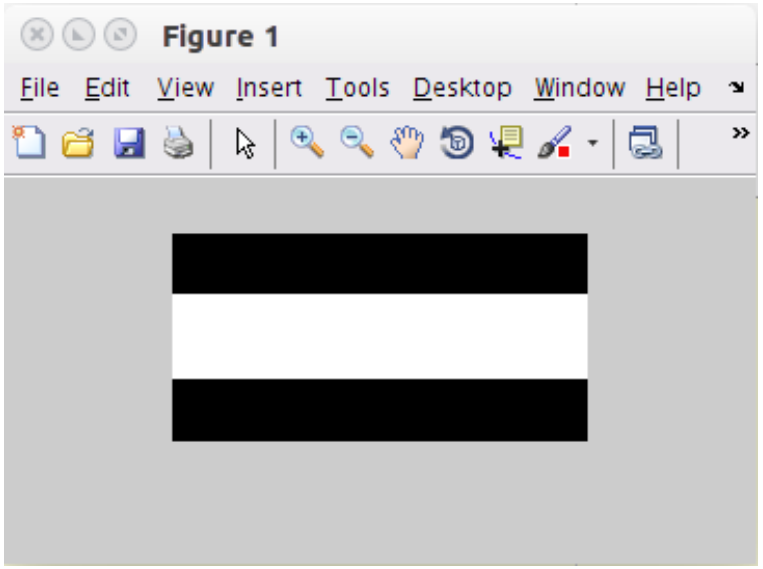


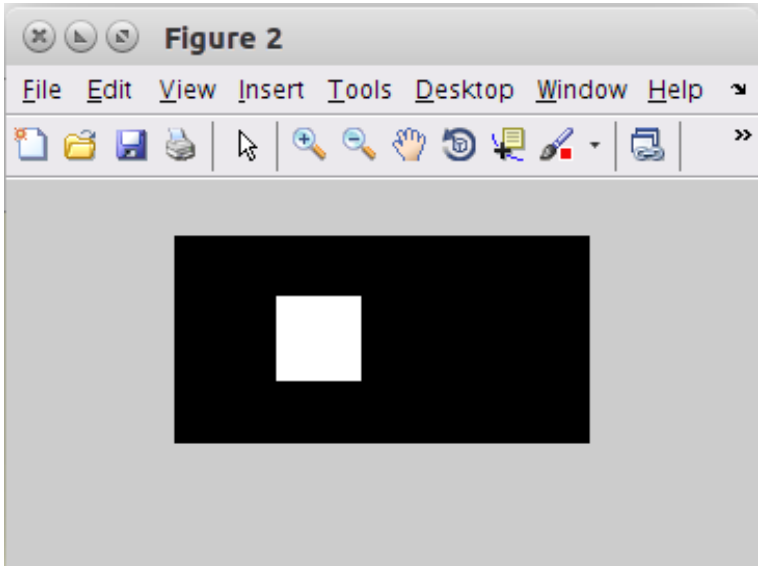
Exercício 1b)

- Inserir um quadrado branco a iniciar na linha 30, coluna 50, com 41 pixels de lado;
- Explorar as possibilidades de indexação do matlab:
  - Primeiro criar apenas uma faixa branca horizontal;
  - Depois, fazer o exercício completo.

```
%% Ex1b
Z=zeros(100,200);
Z(30:70,:)=255;
figure(1)
imshow(Z)

Z=zeros(100,200);
Z(30:70, 50:90) = 255;
figure(2)
imshow(Z)
```





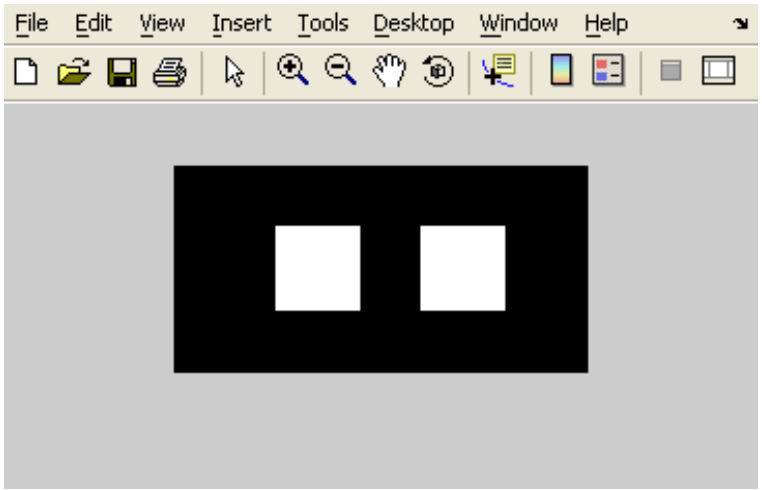
Ex. 2a) - Inserir um quadrado numa imagem vazia

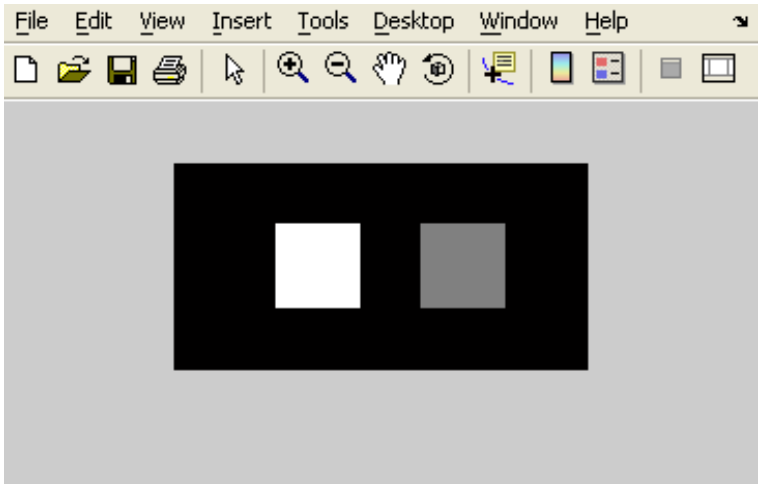
- Na imagem do exercício anterior, colocar um quadrado cinzento (meio da escala =128) ao lado do quadrado branco. O código sugerido parece ser:

```
%% Ex2a

Z=zeros(100,200);
Z(30:70, 50:90) = 255;
Z(30:70, 120:160) = 128;
imshow(Z)
```

- Porque não funcionou e deu a imagem de cima em vez da imagem de baixo como se poderia esperar?
  - Pista para a resposta: tipos de dados (double vs. uint8)





Ex. 2b) - Influência dos tipos de dados

- Repetir o exercício anterior de duas formas:
  - Usando 'doubles'

```
%% Ex 2b
Z=zeros(100,200);  %gera doubles
Z(30:70, 50:90)=1;  %porque^?
Z(30:70, 120:160)=0.5; %porque^?
imshow(Z)
```

-
  - Usando 'uint8'

```
Z=zeros(100,200,'uint8')); %gera uint8 (0-255)
%ou em alternativa: Z=uint8(zeros(100,200)))
Z(30:70, 50:90) = 255;
Z(30:70, 120:160) = 128;
imshow(Z)
```

- Os tipos de dados de variáveis podem-se verificar com o comando `whos` na linha de comando. Por exemplo para ver qual o tipo da variável Z pode-se fazer:
  - whos Z

Nota/Recomendação sobre os tipos de dados

- Para evitar algumas ambiguidades e uniformizar o código futuro, assumir-se-á que as imagens em níveis de cinzento estarão na escala de 0 a 1 representadas por 'doubles'.
- Desvantagens:
  - Trabalhar com 'doubles' é computacionalmente mais exigente do que trabalhar com 'uint8' ou 'uint16'
  - As imagens carregadas de disco (imread()), em geral, vêm em formato 'uint8' ou eventualmente 'uint16' (é preciso converter: im2double())
- Vantagens:
  - A escala 0 a 1 é mais universal, e não é preciso estar a pensar em frações de 256 (uint8) ou de 65536 (uint16) para os níveis de cinzento!
  - A escala 0 a 1 coincide com a escala de imagens binárias com os elementos 'logical' (0=preto e 1=branco), que serão muitos usadas como 'máscaras' em trabalhos futuros da disciplina.
  - Alguns funções do Matlab (zeros(), ones(), etc.) devolvem matrizes de 'doubles' dispensado outras conversões.

Exercício 3a) – Criação de uma função

- Criar uma função para inserir um quadrado com 10 de lado numa dada imagem passada como argumento
  - Nesta fase vamos criar um ficheiro distinto para cada função a desenvolver.
  - O ficheiro tem de ter o mesmo nome que a função mas acrescido da extensão '.m'
- Criar ficheiro AddSquare.m para conter a função a criar e inserir o texto seguinte:

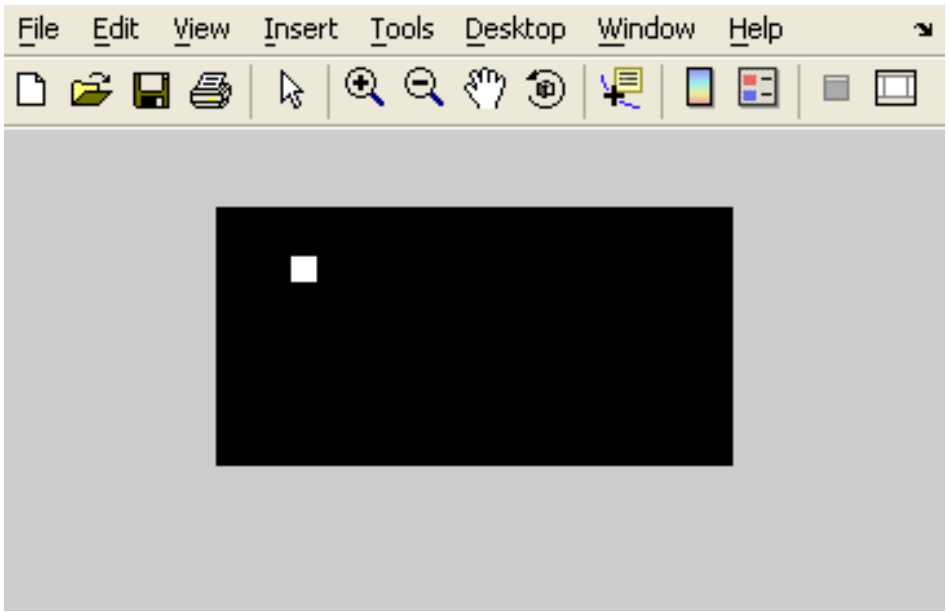
```
function  X=AddSquare ( I, line, col)
%function to add a white square with dimensions (line,col) to image I
I(line:line+10-1, col:col+10-1) = 1;
X=I;
```

- Qual a importância e significado dos '**parâmetros de entrada**' e do '**retorno**' das funções?

Exercício 3b) – Invocação de uma função

- No programa principal invocar a função recém criada:

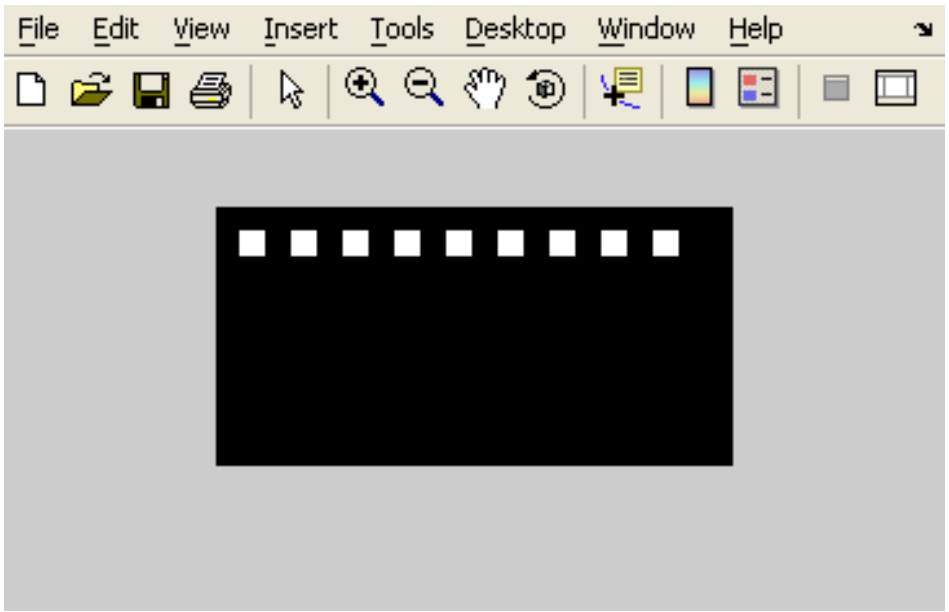
```
%% Ex 3b
clear      %clear all variables
close all  %close all windows, if any
Z=zeros(100,200);
Z=AddSquare(Z, 20,30);
imshow(Z);
```



Exercício 3c) – Uso do ciclo for-end

- Criar um ciclo para gerar múltiplos quadrados (de 20 em 20 pixels):

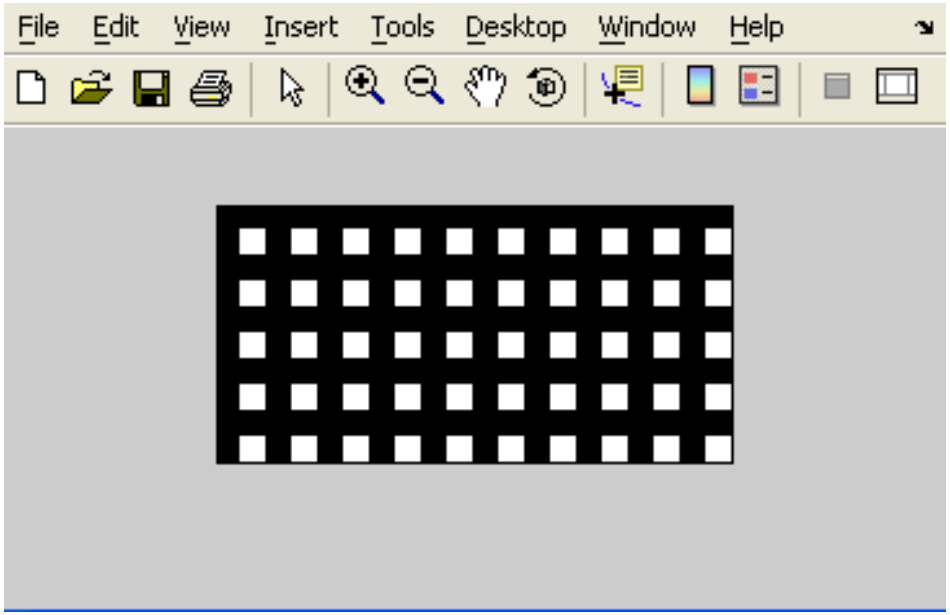
```
%% Ex 3c
Z=zeros(100,200);
for cc=10:20:180                    %Explain the numbers!
      Z=AddSquare(Z, 10,cc);
end
imshow(Z)
```



Exercício 3d) – Um ciclo mais complexo

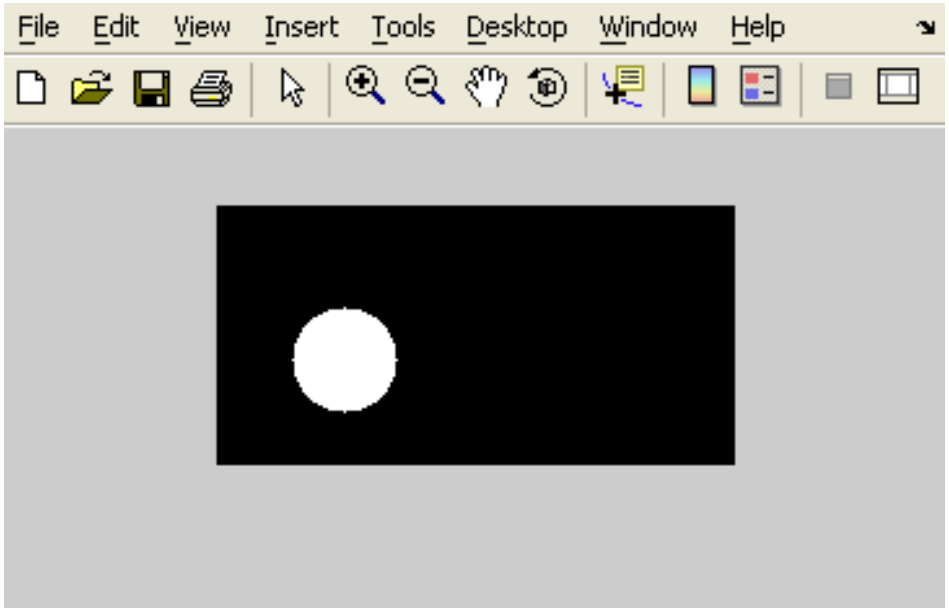
- Encher agora a imagem nas duas dimensões:

```
%% Ex 3d
Z=zeros(100,200);
for ll=10:20:100
    for cc=10:20:200
        Z=AddSquare(Z, ll, cc);
    end
end
imshow(Z)
```



Ex. 4) - Desafio: como criar círculos?

- Como se pode criar um círculo?
  - Pela técnica da indexação não é trivial!
  - Tem de se recorrer a outras técnicas.
  - O Matlab permite-o de um forma alternativa muito elegante.



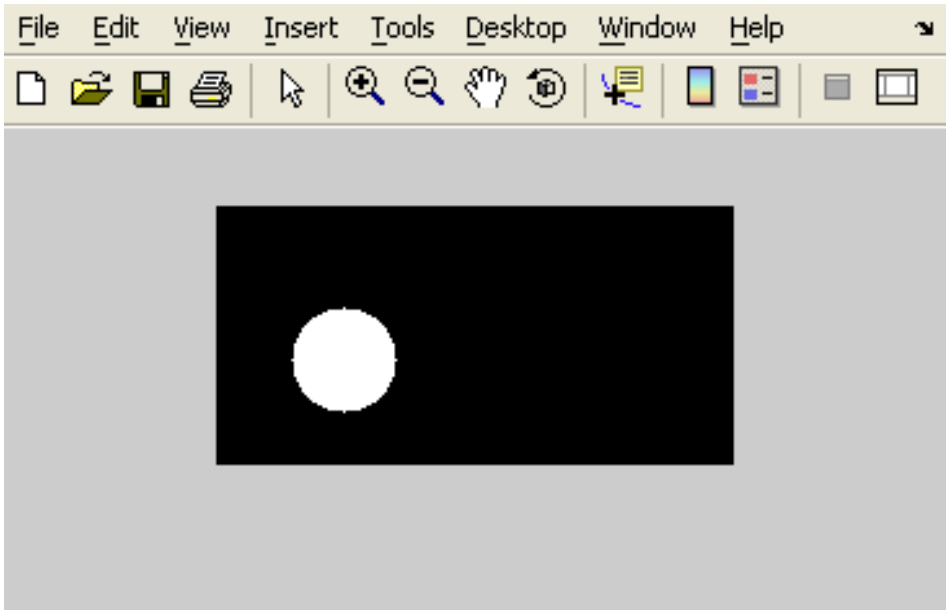
Ex. 4a) - Abordagem da geometria analítica

- Equação de um círculo com centro em (x_0, y_0) e raio r:

$$(x - x_0)^2 + (y - y_0)^2 \leq r^2$$

- Ideia: testar todos os pontos (x,y) da imagem (matriz) e verificar se respeitam a expressão matemática. Usar (x_0,y_0)= (50,60) e r=20.

```
%% Ex 4a
Z=zeros(100,200);
x_0=50; y_0=60; r=20;
for x=1:200
    for y=1:100
        if (x-x_0)^2+(y-y_0)^2 <= r^2
            Z(y,x)=1;
        end
    end
end
imshow(Z)
```



Abordagem matricial para o exercício

- A abordagem anterior pode ser tornada mais eficiente e mais fluida de formular.
- A ideia é a mesma, mas a operação de teste dos pontos deve ser feita toda numa única operação matricial em vez de fazer dois ciclos explícitos.
- Para isso, será preciso ter-se as matrizes com as coordenadas de todos os pontos na imagem para fazer as contas todas de uma vez.

As matrizes de coordenadas (X,Y)

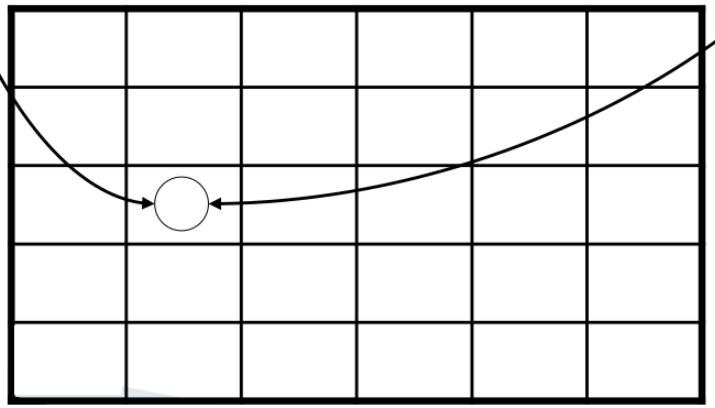
- As duas matrizes (X,Y) com as coordenadas (x,y) (coluna, linha) de todos os pontos da imagem

X=

1	2	3	4	...	M
1	2	3	4	...	M
1	2	3	4	...	M
...	...	...	...	...	...
1	2	3	4	...	M

Y=

1	1	1	1	...	1
2	2	2	2	...	2
3	3	3	3	...	3
...	...	...	...	...	...
N	N	N	N	...	N



Imagem

Ponto (2,3) na imagem

Como construir as matrizes de coordenadas (X,Y)

- As duas matrizes (X,Y) com as coordenadas (x,y) (coluna, linha) de todos os pontos da imagem
- O comando `meshgrid` do matlab
  - Como funciona? Dois vetores permitem gerar duas matrizes.
- Ilustração para x=1:4 e y=1:5.
  - Executar os seguintes comandos na linha de comandos:

```
x=1:4
y=1:5
[X,Y]=meshgrid(x,y)
```

Resultado da operação anterior

```
x =
     1     2     3     4

y =
     1     2     3     4     5

X =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4

Y =
     1     1     1     1
     2     2     2     2
     3     3     3     3
     4     4     4     4
     5     5     5     5
```

As matrizes de coordenadas X,Y da imagem em uso

- Repetir o exercício usando agora as dimensões reais (usar ';' para evitar o *echo* desnecessário na consola)

```
x=1:200; %porque^ 200 e nao 100?
y=1:100; %porque^ 100 e nao 200?
[X,Y]=meshgrid(x,y); %gera matrizes
```

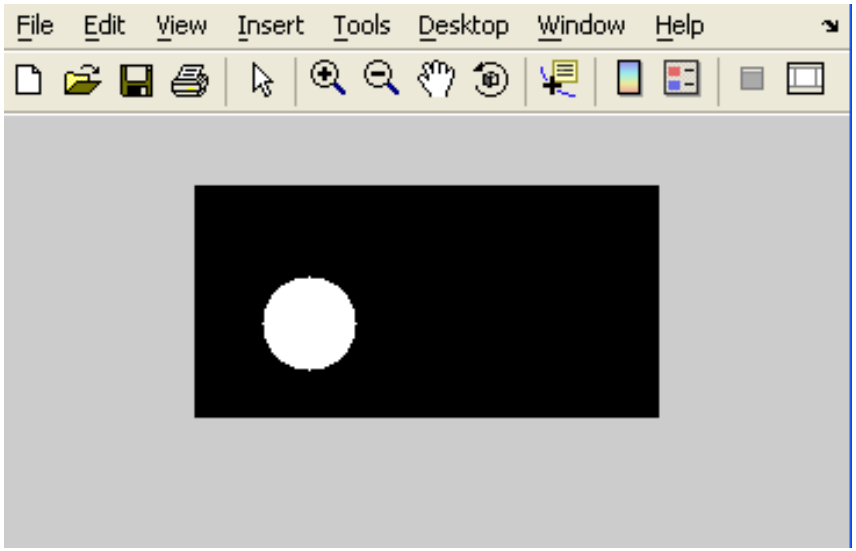
Ex. 4b)-Como utilizar as matrizes X,Y obtidas?

- Resultado de operação lógica (0 ou 1):

```
C=(((X-50).^2 + (Y-60).^2 ) <= 20^2);
%Atencao aos '.' na expressao. Explicar!
```

- Resultado de operação lógica: 0 ou 1
  - O resultado da comparação (<=) gera valores lógicos 0 ou 1 conforme o caso, e é colocado em C (matriz de igual dimensão às de X e Y)

```
%% Ex 4b
close all
Z=zeros(100,200);
x_0=50; y_0=60; r=20;
x=1:size(Z,2);
y=1:size(Z,1);
[X,Y]=meshgrid(x,y);
C=(((X-x_0).^2 + (Y-y_0).^2 ) <= r*r);
Z(C)=1;
imshow(Z)
```



- Explicar a expressão: Z(C)=1

Explicação de `Z(C) = a`

- Sendo:
  - Z uma matriz qualquer
  - C uma matriz booleana de dimensões iguais a Z
  - a um escalar qualquer
- `Z(C) = a` é equivalente ao seguinte código matlab:

```
for x=1:size(Z,2)
    for y=1:size(Z,1)
        if C(x,y) == 1 %True
            Z(x,y) = a;
        end
    end
end
```

Ex. 4c) – Criar uma função para adicionar círculos a uma imagem

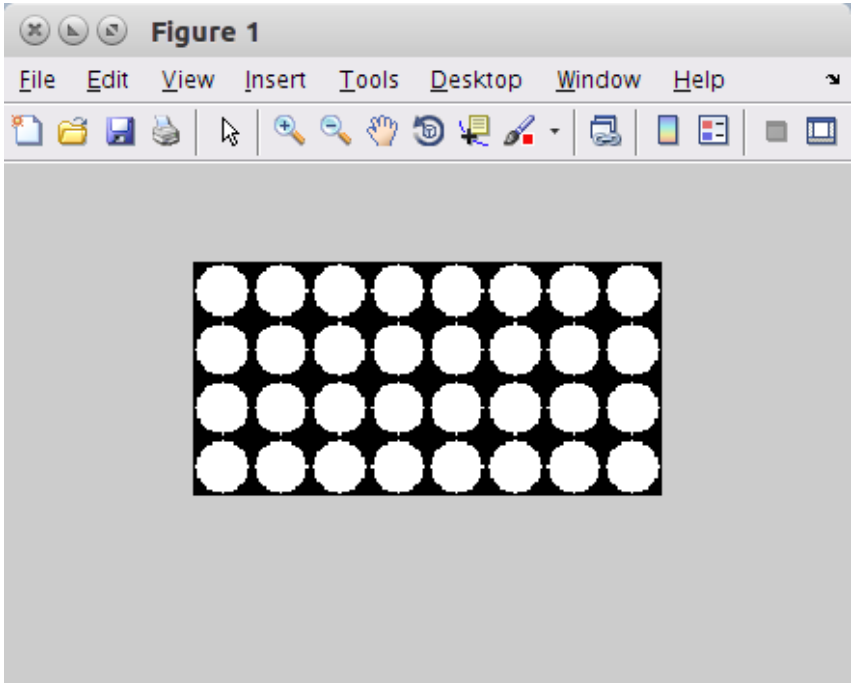
- Que parâmetros deve ter a função? O que são parâmetros?
- Como adicionar à imagem passada?
- Como fazer a função independente da imagem passada?
- Explicar os parâmetros e o valor de retorno ...

```
function I=AddCircle(Z, x0, y0, r)
%Function to create a white circle in Z with center in x0,y0 and radius r
x=1:size(Z,2);
y=1:size(Z,1);
[X,Y]=meshgrid(x,y);
C=(((X-x0).^2 + (Y-y0).^2 ) <= r*r);
I=Z;
I(C)=1;
```

Ex. 4d) – Criar uma imagem com vários círculos (ciclos for)

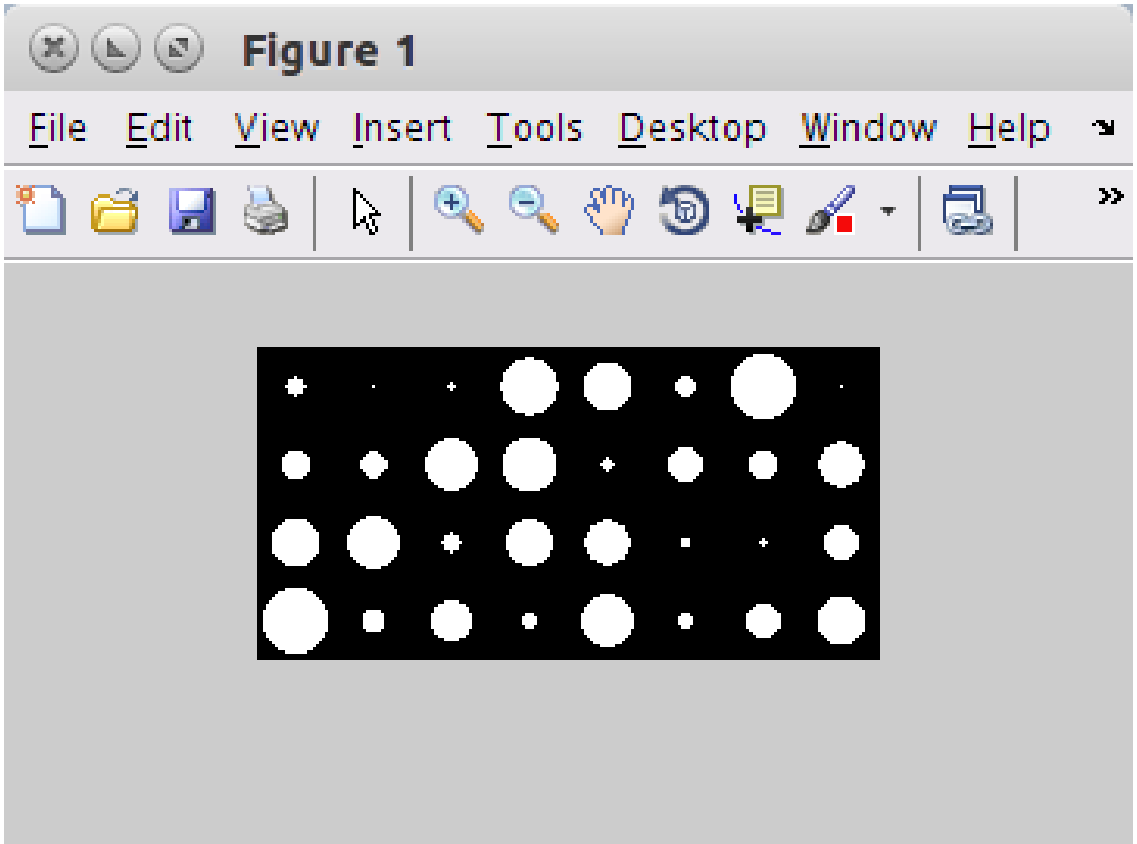
- Usando a função `AddCircle`, preencher uma imagem preta com círculos brancos como ilustrado.

```
%% Ex 4d
close all
Z=zeros(100,200);
for ll=13:25:100
    for cc=13:25:200
        Z=AddCircle(Z, cc, ll, 11 );
    end
end
imshow(Z)
```



Ex. 4e) – Criar uma imagem com círculos de raio aleatório

- Adaptar o exercício anterior para gerar círculos com raio aleatório. Sugestão: usar a função rand()



Ex. 5) – Uma imagem para S. Valentim (op.)

- Adaptar o exercício de inserção do círculo para um "coração":
- Equação de um coração com centro em (x_0, y_0):

$$\left[(x - x_0)^2 + (y - y_0)^2 - 1\right]^3 - (x - x_0)^2(y - y_0)^3 \leq 0$$

- Nota: a escala fica muito pequena para vizualização da imagem em unidades inteiras das coordenadas. Sugere-se dividir os valores de x e y por um fator (por exemplo, 30) para aplicar na equação anterior.
  - Adicionalmente, o "coração" surge invertido na imagem por causa do sentido de crescimento da variável Y. Recomenda-se uma operação de `flipud()` para ter o coração na posição não invertida, como ilustrado.