

# Sistemas de Visão e Percepção Industrial

## Aula Prática nº 3

Transformações de imagens em matlab.

Transformação de perspetiva.

# Uma função para transformar imagens

- `g=imwarp()` – função principal
- Argumentos obrigatórios:
  - `warp(A, tform)`
    - `A` - original image
    - `tform` – special transformation matrix
  - `tform` obtém-se usando `affine2d()` (mas há outras)
    - usa a matriz de transformação real (mas recorre à transposta na notação usual)
- A função tem outros argumentos possíveis (opcionais)
- `warp(A, Ri, tform, 'OutputView', Ro, ... )`
  - `Ri` - sistema de referência da imagem de entrada (obtido com `imref2d(size(A), xlimits, ylimits)` )
  - `'OutputView', Ro` - par de parâmetros para alterar o espaço de referência da imagem de saída (`Ro` é obtido também com `imref2d(size(OutputImage), outxlimits, outylimits)` ). Os vetores `outxlimits` e `outylimits` têm um valor por defeito de `[0 D]`, onde `D` é a dimensão respetiva da imagem.
  - Jogar com os limites e as dimensões nos sistemas de referência das imagens de entrada ou saída afetará a escala da imagem resultante.

# Exercício 1) Rotação de uma imagem

- Fazer a rotação da imagem 'rice.png' de  $\pi/4$  ( $45^\circ$ ) por duas formas diferentes: `imrotate()` e `imwarp()`

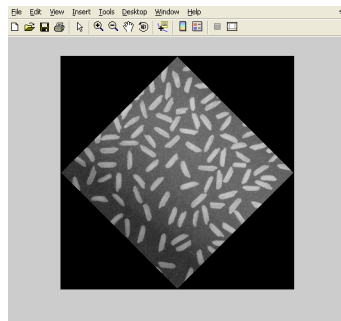
```
Z=im2double(imread('rice.png'));
a=pi/4;

%Duas soluções diferentes de fazer uma ação semelhante

%=====Solução 1 - específica =====
newZ2=imrotate(Z,-a*180/pi);
figure(1); imshow(newZ2);

%=====Solução 2 - abordagem geral =====
T=rot(a); %função criada na aula 2
%Criar a matriz de transformação
tf = affine2d(T');

% aplicar a matriz de transformação
newZ1=imwarp(Z, tf);
figure(2); imshow(newZ1)
```



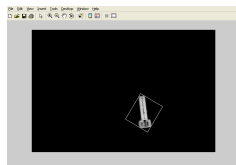
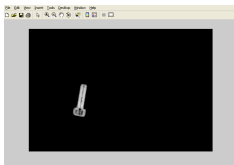
- NB. A função `affine2d()` usa a matriz de transformação na forma transposta.

## Exercício 2a) Transformação geral de imagem

- Transformar a imagem 'bolt1.png' de forma a que apareça com translação e orientação aleatória numa imagem de  $400 \times 600$ 
  - Usar a transformação  $T = \text{trans}(x,y) * \text{rot}(a)$ , com  $x$ ,  $y$  e  $a$  aleatórios.
  - Com `imref2d()`, criar a referência para a imagem de saída com as dimensões pedidas.
  - Sugestão: em `imwarp()`, usar o parâmetro 'SmoothEdges' como **true** para suavizar linhas de contorno (embora seja opcional).

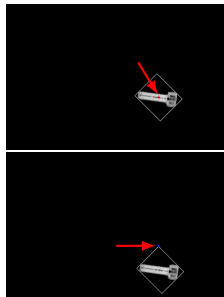
```
cols=600; lins=400;  
T=trans(x,y)*rot(a);  
tf=affine2d(T);  
%Default: object origin is top left  
Ro = imref2d([lins cols]);  
tempA=imwarp(A,tf,'OutputView',Ro,'SmoothEdges',true);
```

- Para melhor visualizar o efeito, forçar um bordo branco na imagem original antes de aplicar a transformação. Como se pode fazer?




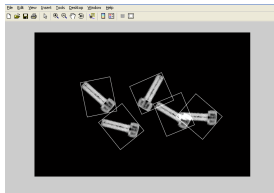
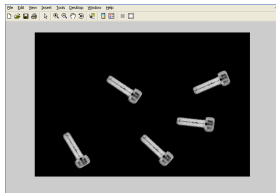
## Exercício 2b) Outros parâmetros de `imwarp`

- Repetir o exercício anterior, mas agora usar o centro do objeto original como o centro de rotação. Usar um parâmetro para a referência da imagem de entrada em `imwarp()` para esse efeito.
- Usar a transformação  $T = \text{trans}(x, y) * \text{rot}(a)$ , com  $x$ ,  $y$  e  $a$  aleatórios.
- Usar o parâmetro para  $R_i$  para referência de 'input' na função `imwarp()`:
  - `wObj=size(A,2); hObj=size(A,1);`
  - `imxlim=[-wObj/2 wObj/2];`
  - `imylim=[-hObj/2 hObj/2];`
  - `Ri=imref2d(size(A),imxlim,imylim);`
  - `B=imwarp(A, Ri, tf,...);`
- Comparar os efeitos com e sem o parâmetro  $R_i$ ; Criar duas imagens, uma para cada caso.
- O parâmetro e argumento 'Interp', 'nearest' da função `imwarp()` também pode ser útil no resultado da imagem final.



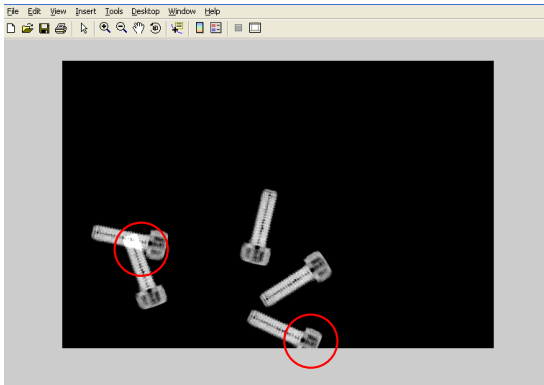
## Exercício 2c) Combinação de imagens

- Gerar uma imagem composta de 5 imagens de 'bolt.png', mas todas com rotações e translações aleatórias.
- A imagem final deve ser uma "combinação" das 5 imagens, e isso pode ser feito de várias formas:
  - Soma das 5 imagens ( pode gerar pixels com valores  $> 1$ ):
    - `accumul = accumul + singleBolt;`
  - A nova imagem deve ser o máximo da anterior e da imagem parcela:
    - `accumul = max(accumul, singleBolt); %Explicar`
  - Substituição sucessiva na imagem acumulada (`accumul`) dos pixels não pretos da imagem parcela (`singleBolt`):
    - `mask=(singleBolt>0); accumul(mask)=singleBolt(mask);`
- Exemplos das imagens combinadas: sem bordo e com bordo



## Exercício 2d) Composição sem sobreposição

- No exercício anterior, como se pode fazer para garantir que na imagem final sintetizada não há sobreposição de objetos (parafusos) nem objetos sobre os bordos da imagem?
  - Sugestão: uma nova imagem não pode ser "adicionada" à imagem global se tiver pixels com valor maior do que 0 em comum.
  - No caso do bordo, esse teste será feito com um caixilho branco!



## Exercício 3a) Gerador de imagens de dominós (opcional\*\*)

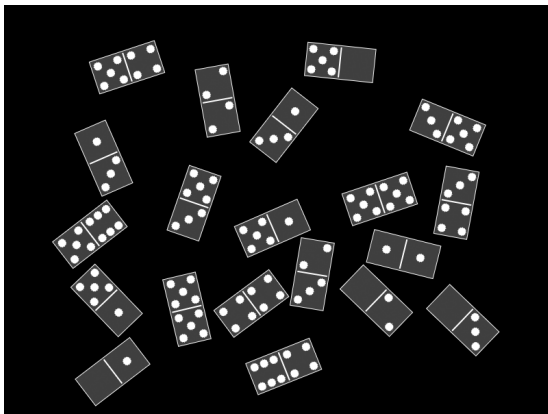
- Criar uma função que retorne uma imagem de um dominó e aceite dois argumentos que são o número de pintas em cada metade.
  - As dimensões recomendadas são 50 linhas e 101 colunas.
  - A função deve impor um número de pintas como inteiro de 0 a 6.
  - O corpo da peça do dominó deve ser cinzento a 25%, e as pintas brancas circulares de raio 5.
  - A peça deve ter um bordo branco de um pixel à sua volta, e deve ter uma linha separadora central de 2 pixels de largura e 44 de altura.
  - Admitindo que a função se designa `create_domino`, chamadas com os seguintes formatos devem devolver imagens como as ilustradas:
    - `A=create_domino(2,6); B=create_domino(0,4); C=create_domino(3,3);`





## Exercício 3b) Dominós aleatórios (opcional)

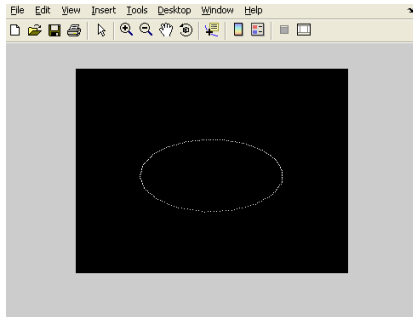
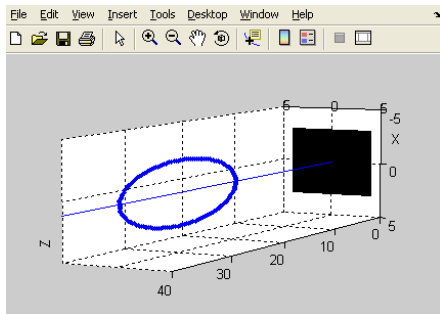
- Recorrendo à função do exercício anterior, criar um programa para gerar imagens de 600x800 com até 20 dominós de valores aleatórios que não se tocam nem tocam no bordo.



- Como se poderia fazer para garantir que não há dominós repetidos?

## Exercício 4) Transformação de perspectiva

- Testar e verificar a transformação de perspectiva para um dado conjunto de pontos a 3D.
- Ilustra-se uma possibilidade de resultado esperado



## Exercício 4) Decomposição do problema

Passos para o exercício

- ➊ Definição do espaço para a imagem final
  - Imagem inicializada a preto (zeros) (ex. 240 X 320)
- ➋ Criação de um objeto no espaço a 3D
  - Exemplo de pontos sobre uma circunferência (eg. 50 pontos)
- ➌ Homogeneização de coordenadas
  - Acrescentar a linha de "1s"
- ➍ Eventuais transformações geométricas do objeto no espaço
  - Alterar o objeto com rotações e translações
- ➎ Representação do objeto em 3D (com plot ou fill)
  - plot é suficiente para ver pontos (fill destaca mais o objeto)
- ➏ Definição da matriz intrínseca da câmara
  - Criar a matriz com os parâmetros adequados
- ➐ A transformação de perspectiva/projeção
  - Aplicar a matriz sobre o objeto e des-homogeneizar
- ➑ Visualizar a imagem projetada
  - Testes para excluir pontos que se projetam fora da imagem
  - Alguns "truques" dado que o Matlab não permite todas as indexações

## Exercício 4) Fase preparatória

- Primeiro, criar 4 funções para as transformações geométricas no espaço a 3D e colocá-las na pasta 'lib'
  - $M = \text{trans3}(x, y, z)$
  - $M = \text{rotx}(a)$
  - $M = \text{roty}(a)$
  - $M = \text{rotz}(a)$

$$\text{trans3}(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{rotx}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$
$$\text{roty}(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{rotz}(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Exercício 4a)

- Criar a matriz para a imagem
- Criar o objeto circular

```
%Definição do espaço para a imagem final
imLins=240;
imCols=320;
A=zeros(imLins, imCols);

%Criação de um objeto no espaço a 3D
%exemplo de um círculo com 50 pontos
t=linspace(0, 2*pi, 50);
P=[ 5*cos(t);    %explicar!
    5*sin(t);    %explicar!
    30*ones(size(t))]; %explicar!
```

## Exercício 4b)

- Homogeneizar as coordenadas do objeto
- Aplicar transformações geométricas adequadas para colocar o objeto numa dada posição no espaço (rotações e translações).

```
%Homogeneização de coordenadas
```

```
P = [P; ones(1,size(P,2))];
```

```
%Transformações geométricas do objeto no espaço
```

```
%Atenção à ordem e significado
```

```
P=trans3(0,0,30)*rotx(0)*roty(0)*trans3(0,0,-30)*P;
```

```
%Podem usar-se outros valores de rotação e translação!
```

## Exercício 4c)

- Representar o objeto a 3D.
- Ajustar as condições de visualização.
- Representar o eixo ótico e o plano da imagem.

```
%Representação do objeto (com plot ou fill)
%fill3(P(1,:), P(2,:), P(3,:), 'm');
plot3(P(1,:), P(2,:), P(3,:), '.');
hold on; grid on; axis equal;
axis([-5 5 -5 5 0 40]);
xlabel('X'); ylabel('Y');
line([0 0], [0 0], [0 50]); %0 eixo Z
fill3([4 -4 -4 4], [-3 -3 3 3], [0 0 0 0], 'k'); % :-)
```

## Exercício 4d)

- Obter a matriz intrínseca da câmara criando a função `PerspectiveTransform()`

```
%Definição da matriz intrínseca da câmara  
alpha=[500 500]; %% fixed for a given camera  
center = [size(A,2) size(A,1)]/2;  
K = PerspectiveTransform( alpha, center);
```

```
function K=PerspectiveTransform(alpha, center)  
K=[alpha(1) 0      center(1) 0  
    0      alpha(2) center(2) 0  
    0      0      1      0];
```



## Exercício 4e)

- Aplicar a transformação de perspectiva e des-homogeneizar.

```
%%A transformação de perspectiva/projecção  
Ch = K*P;  
%Convert homogeneous to non-homogeneous  
C = round( Ch(1:2,:) ./ repmat(Ch(3,:),2,1) );  
%because y (lines) starts on top of image  
C(2,:) = size(A,1) - C(2,:); %Explain!
```

- Nesta fase, C já contém o resultado da projecção... será a imagem final...? Não, são só as coordenadas onde houve projecção dos pontos reais...! Agora é preciso criar uma imagem onde essas coordenadas estarão com pixel visível!

## Exercício 4f)

- Excluir de C os índices que estão para além dos limites antes de criar uma imagem com pixels nessas coordenadas!
- Coordenadas devem ser  $> 0$  e menores que os limites da imagem, sob risco de erros.

```
%Tests to exclude out of range pixels....  
OKs=( C(2,:)>0 & C(2,:)<=imLins) & ( C(1,:)>0 & C(1,:)<=imCols ) ;  
%OKs is linear array with 0s and 1s. Can be used as indexers.  
C2=C(2,OKs);  
C1=C(1,OKs);  
C=[C1;C2]; %C is now the set of coordinates with "white" pixels.  
~
```

- Quais as dimensões da variável OKs?
- No fim desta operação, C já só tem valores de coordenadas que estão dentro dos limites da imagem definida.

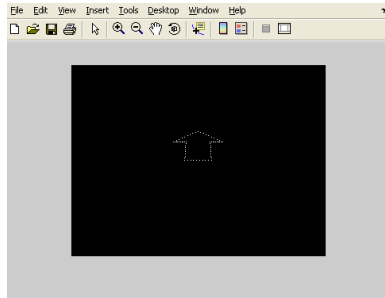
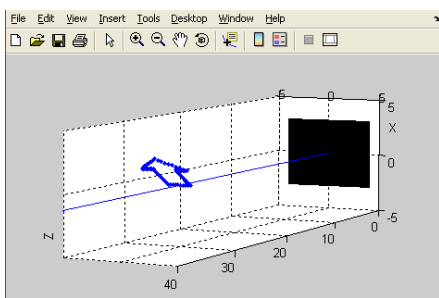
## Exercício 4g)

- Sintetizar a imagem projetada
- O matlab obriga a linearizar as coordenadas porque, por exemplo  $A(C)=1$  não funciona!
- Assim, a solução é converter as coordenadas x,y em índices lineares absolutos mediante a função `sub2ind()`

```
%Now a trick since not every indexing works in Matlab :-(  
idx=sub2ind(size(A), C(2,:), C(1,:)); % Convert (y,x) to linear indices  
A(idx)=1; %make those pixels white... they are the projected pixels
```

# Exercício 5) Projetar outros objetos (opcional)

- Repetir o exercício com outros objetos.
- Por exemplo, uma seta em orientações várias.



- Para facilitar a criação de um objeto com pontos ao longo das suas arestas dados os seus vertices, sugere-se criar uma função (`genpts()`) que dados dois pontos (a 3D) gere um conjunto de pontos uniformemente espaçados entre esses dois pontos extremos.
- Por exemplo, se  $A = [0 \ 0 \ 0]'$  e  $B = [3 \ 6 \ 12]'$ ,  $P = \text{genpts}(A, B, 4)$  deve dar:
  - $P = [0 \ 1 \ 2 \ 3; 0 \ 2 \ 4 \ 6; 0 \ 4 \ 8 \ 12]$ .