

VISÃO COMPUTACIONAL - Unimontes

Mestrado em Modelagem Computacional e Sistemas

Professor: **Antônio Wilson Vieira**



Bibliografia

Bibliografia básica:

- TRUCCO, Emanuele; VERRI, Alessandro. Introductory Techniques for 3-D Computer Vision. New York, USA: Prentice Hall, 1998.
- FORSYTH, David A; PONCE, Jean. Computer Vision: A Modern Approach. New York, USA: Prentice Hall, 2002.
- SZELISKI, Richard. Computer Vision: Algorithms and Applications. Springer-Verlag, 2010.

Bibliografia complementar:

- Castleman, K.R. DIGITAL IMAGE PROCESSING, New Jersey (USA), Editor Prentice-Hall, 1996.
- Gonzalez, R.C. DIGITAL IMAGE PROCESSING, 3 rd ed.USA, Editor Addison-Wesley, 1992.
- Hanalick, R.M.Shapiro, L.COMPUTER AND ROBOT VISION, USA, Editor Addison-Wesley, vol. 1, 1991.

Principais Journals

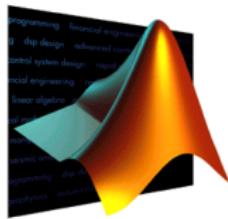
- PAMI - Pattern Analysis and Machine Intelligence
- IJVC - International Journal of Computer Vision
- PRL - Pattern Recognition Letters
- TIP - Transaction on Image Processing
- CVIU - Computer Vision and Image Understanding
- JMIV - Journal of Mathematical Imaging and Vision

Principais Conferências

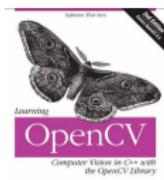
- ICCV - Internation Conference on Computer Vision
- CVPR - Computer Vision and Pattern Recognition
- ICPR - International Conference on Pattern Recognition
- SIBGRAPI - Brazilian Conference on Computer Graphics
- WCV - Brazilian Workshop on Computer Vision

Linguagens e Implementação

- Matlab



- OpenCV em C/C++



* Videoaula EVCOMP 2014: Introdução à Biblioteca de Visão Computacional

Professor

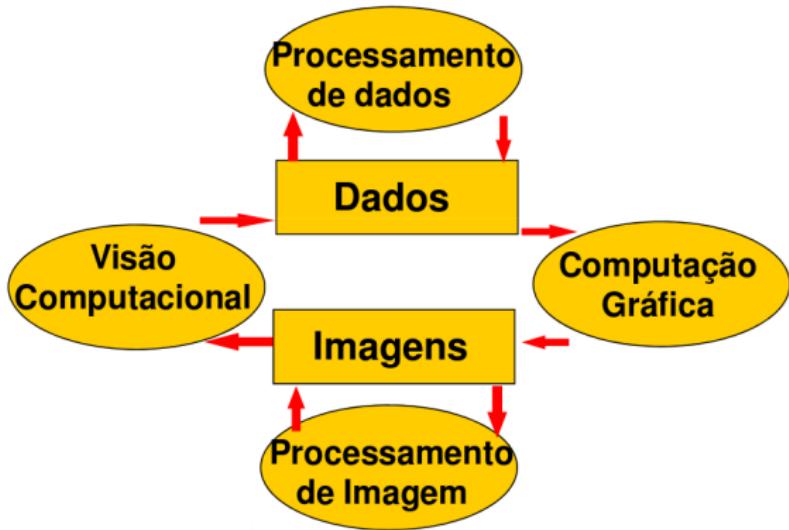
Antônio Wilson Vieira

Sala 14A

Email: *antonio.vieira@unimontes.br*

home: *www.dcc.ufmg.br/~awilson*

Visão Computacional

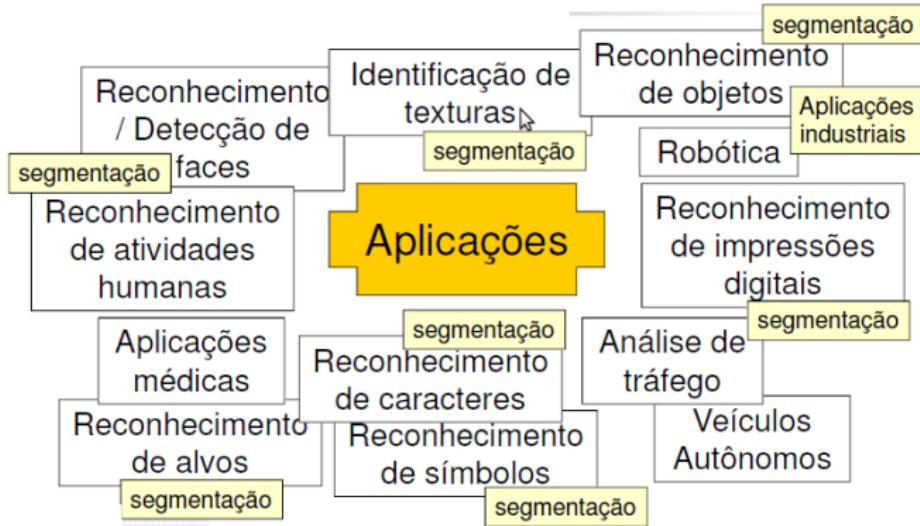


Visão Computacional

Analogia com visão humana

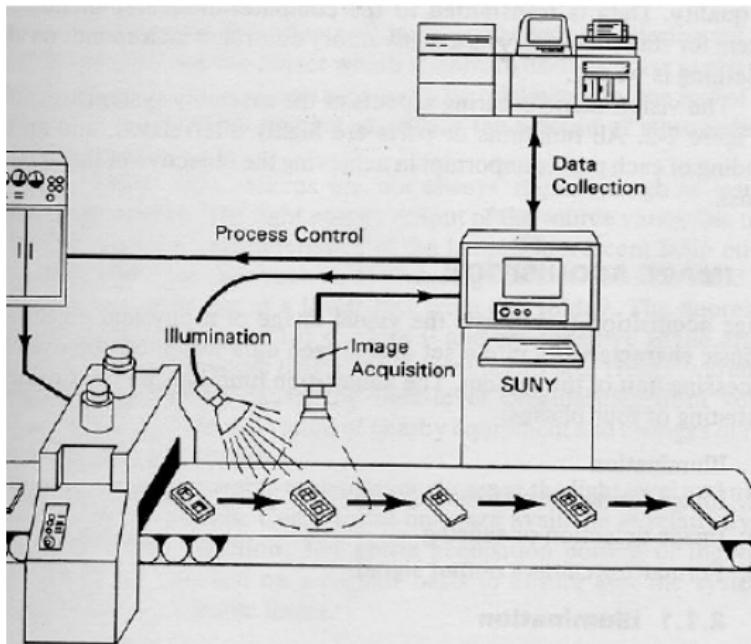


Visão Computacional



Visão Computacional

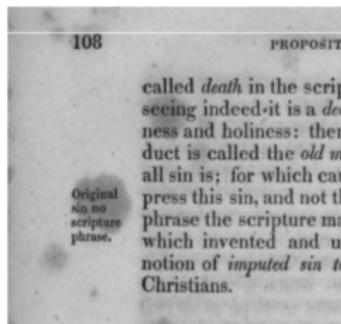
Aplicações Industriais



Visão Computacional

Reconhecimento de texto

RS
781,434



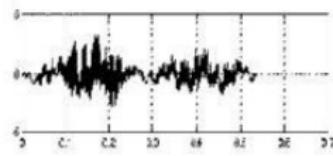
(135,1)

port, fozem da Para e ha
louca, cosa de algum
anans 2000 em Nasc
Outro they came anterior deve
a parte da justica no obvio Bro.
sej republicas proeminentes da
Ria grande e mato em
P. St.

Eu veo tags em ma tag -
nico e a medida tel dor
mato, que é mato
combinação que é mato. Como
Chernomont este é para e'

Visão Computacional

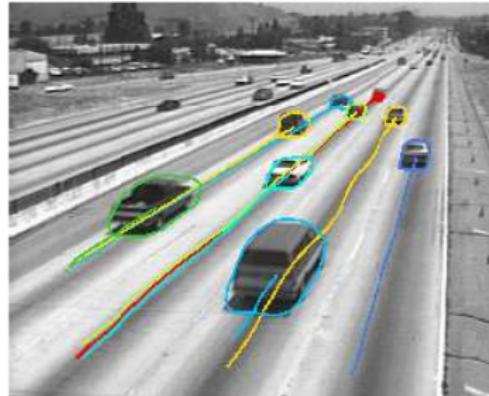
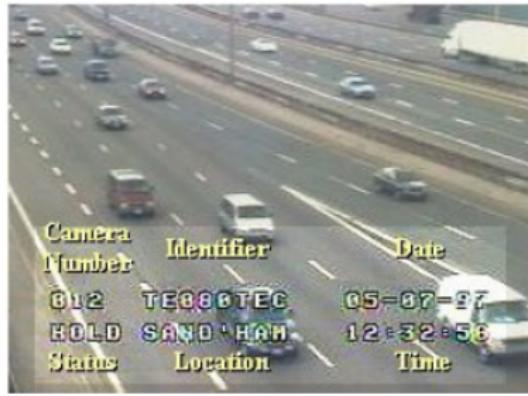
Biometria



John Smith

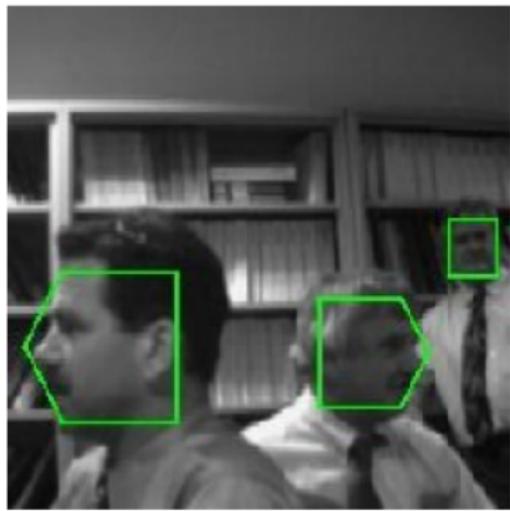
Visão Computacional

Análise de tráfego



Visão Computacional

Detecção de faces



Visão Computacional

Reconhecimento de faces

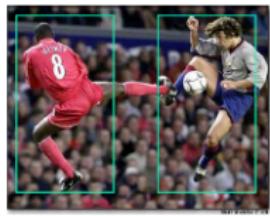


Imagens de treino



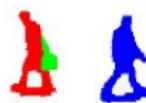
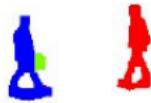
Visão Computacional

Detecção de pessoas



Visão Computacional

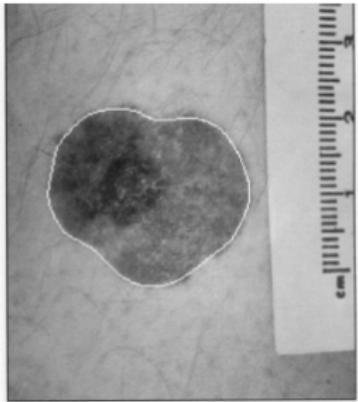
Reconhecimento de ações/atividades humanas



Visão Computacional

Aplicações médicas

Câncer de pele



Câncer de mama



Visão Computacional

Câmeras 3D ou de profundidade

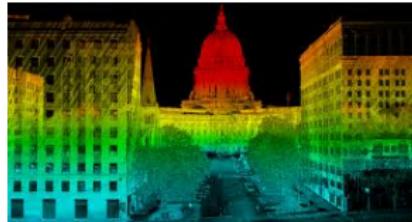


SR4000

CamCube2.0

Kinnect

PR2 Cam



Stuart Miller

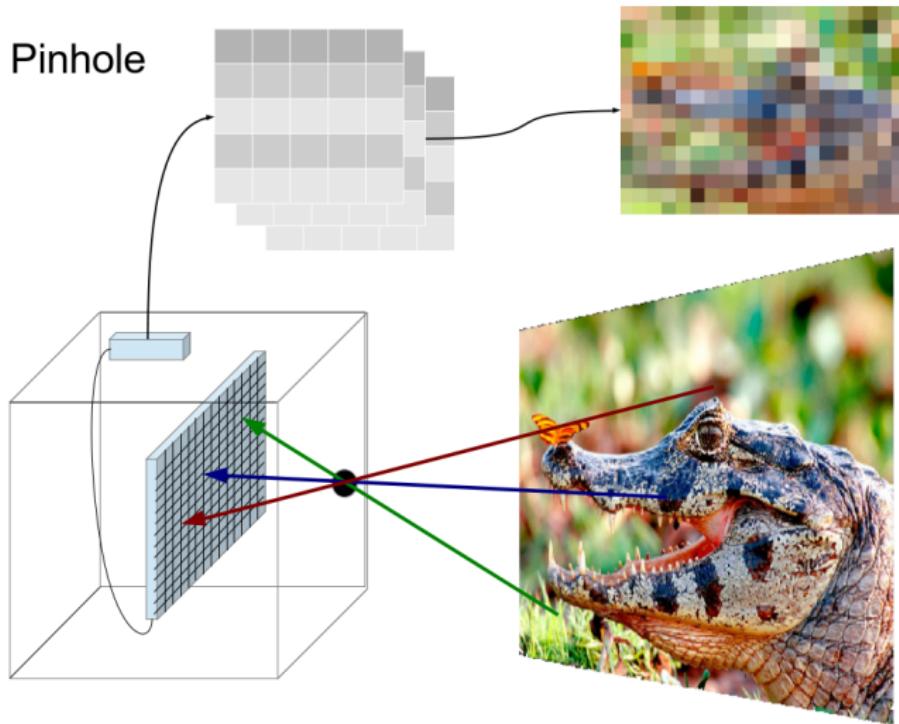


Willow Garage

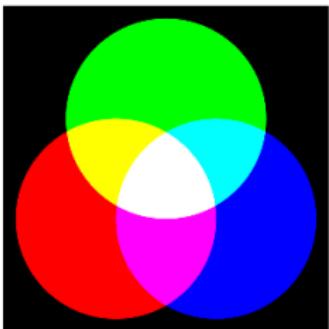


Microsoft

Visão Computacional



Visão Computacional



R	G	B	Cor
0	0	0	Preto
1	0	0	Vermelho
0	1	0	Verde
0	0	1	Azul
1	1	0	Amarelo
1	0	1	Magenta
0	1	1	Cyan
1	1	1	Branco

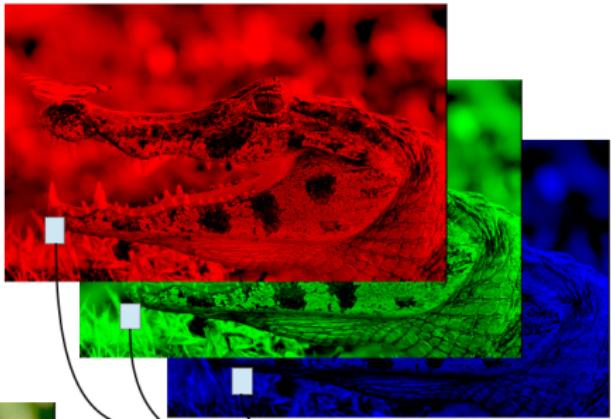
Quantidade de bits	Qtdade de cores
1 bit	2 cores (b / w)
3 bits	8 cores
8 bits	256 cores
24 bits	16.777.216 cores

Visão Computacional

Se uma imagem tem
800 x 600 pixels,
então é armazenada em
800 x 600 x 3 bytes

$$= 1.440.000 \text{ bytes}$$

$$= 1.44 \text{ Mb}$$



R	G	B
1 byte	1 byte	1 byte
8 bit	8 bit	8 bit
0-255	0-255	0-255

24 bits por pixel
3 bytes por pixel

Visão Computacional

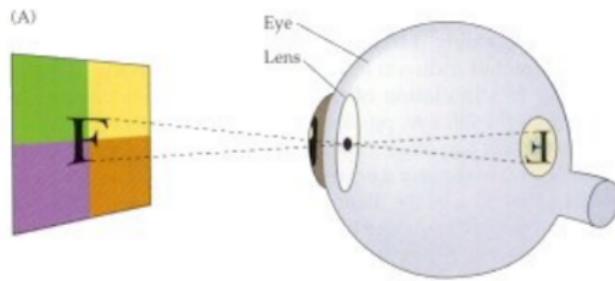
PIXELS



```
136 134 161 159 163 168 171 173 171 166 159 157 155  
152 149 136 130 151 149 151 154 158 161 163 163 159 151  
145 149 149 145 140 133 145 143 145 145 145 146 140 148  
148 143 141 145 145 145 141 136 136 135 135 136 135 133  
131 131 129 129 133 136 140 142 142 138 130 128 126 120  
115 111 108 106 106 110 120 130 137 142 144 141 129 123  
117 109 098 094 094 098 100 110 125 136 141 147 147 145  
136 124 116 105 096 096 100 107 116 131 141 147 150 152  
152 152 137 124 113 108 105 108 117 129 139 150 157 159  
159 157 157 159 159 121 120 120 121 121 121 136 147 158 163  
165 165 163 163 163 166 136 131 135 138 140 145 154 163  
166 165 170 160 166 168 170 173 145 143 147 146 152 159  
166 173 173 175 173 171 170 173 177 178 151 151 153 156  
161 170 176 177 177 179 176 174 174 176 177 179 155 157  
161 162 166 176 180 180 182 180 175 175 176 180 180 180
```

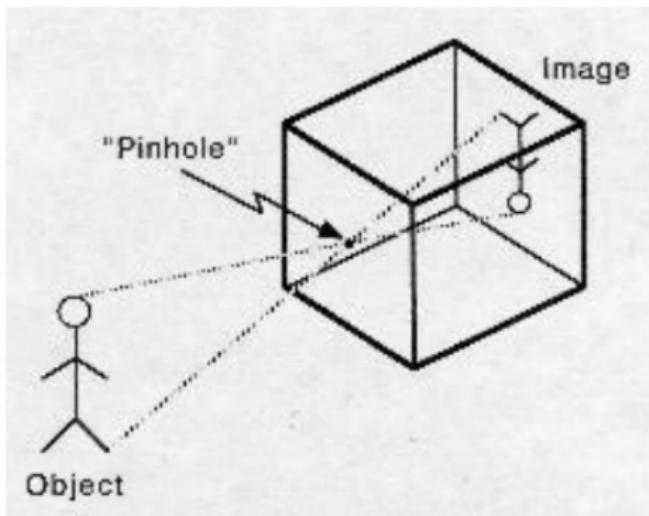
Visão Computacional

Visão humana



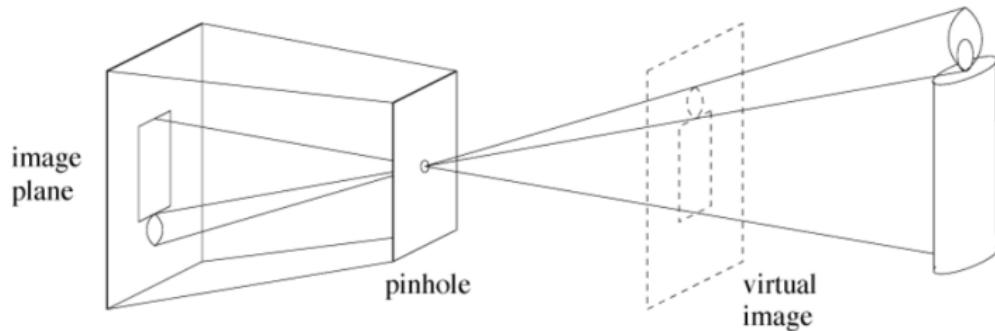
Visão Computacional

Imagen invertida



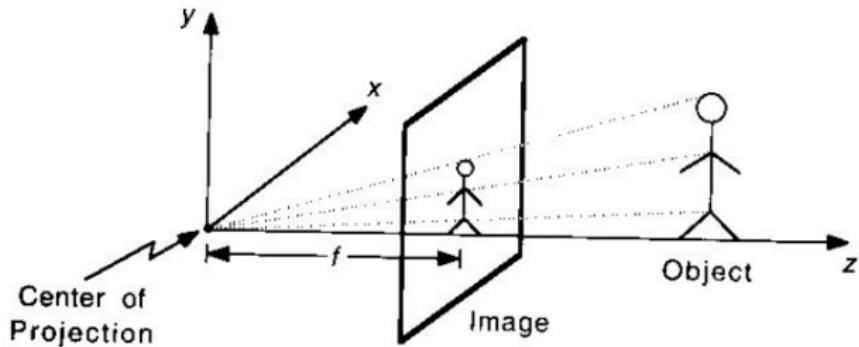
Visão Computacional

Imagen virtual



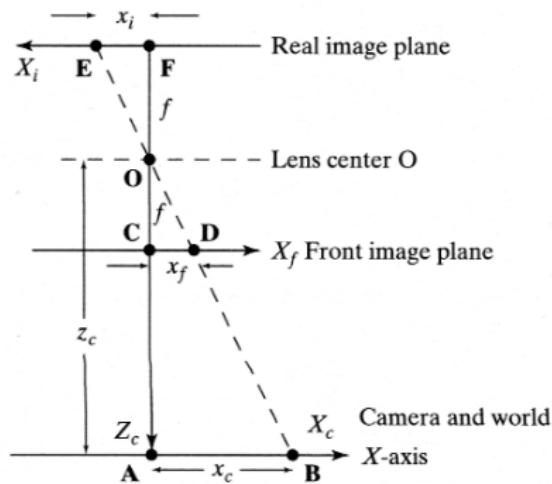
Visão Computacional

Modelo geométrico de câmera sem imagem invertida



Visão Computacional

Imagen 1D

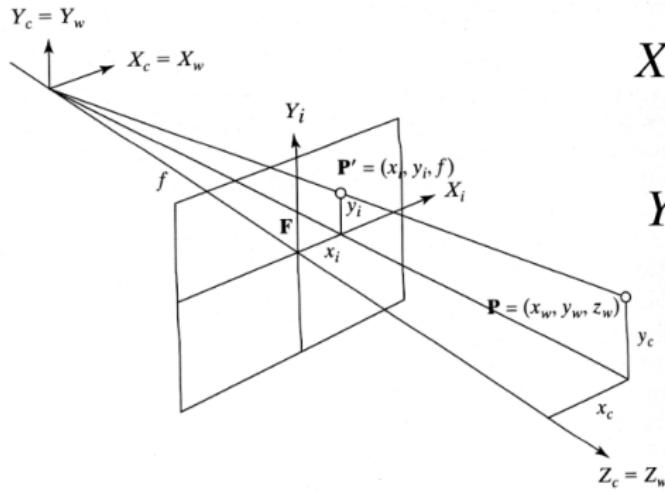


$$X_f / f = X_c / Z_c$$

$$X_f = f \frac{X_c}{Z_c}$$

Visão Computacional

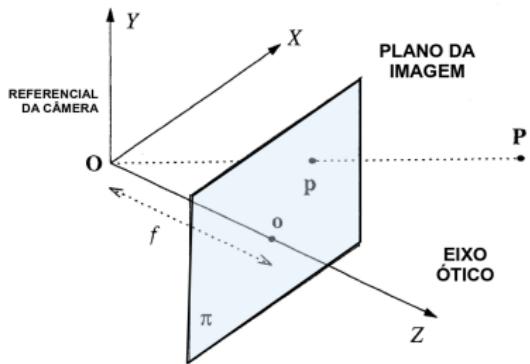
Imagen 2D



$$X_i = f \frac{X_c}{Z_c}$$

$$Y_i = f \frac{Y_c}{Z_c}$$

Câmera perspectiva



O modelo geométrico de câmera perspectiva consiste em um plano π que é o *plano da imagem* e um sistema de coordenadas com origem **O**, chamado *centro de projeção*. A distância do ponto **O** ao plano π , indicada por f , é a *distância focal*.

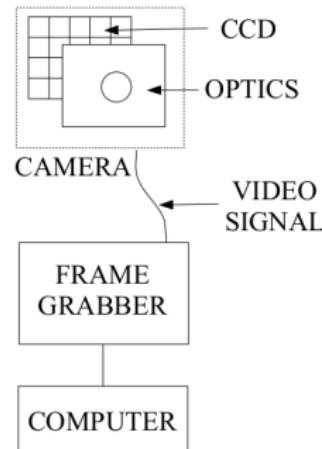
Nesse sistema, dado um ponto $P = [X, Y, Z]^T$ e sua imagem $p = [x, y, z]^t$ no plano π , são imediatas as equações fundamentais

$$x = f \frac{X}{Z} \quad \text{e} \quad y = f \frac{Y}{Z}$$

Formação da imagem

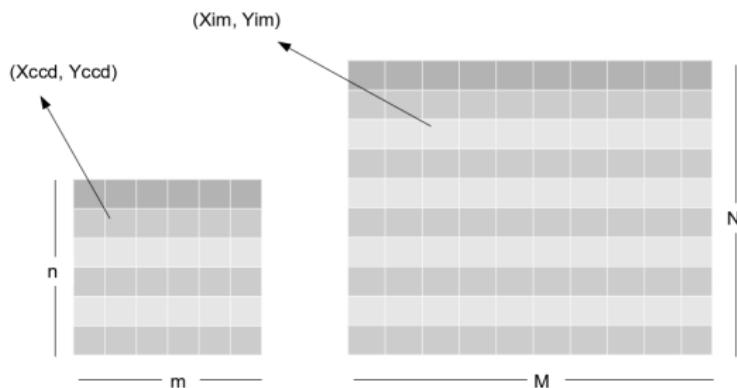
Uma imagem digital é representada por uma matriz numérica, E , com N linhas e M colunas.

$E(i,j)$ é um inteiro no intervalo [0,255] indicando a intensidade no **pixel** (i,j)



Formação da imagem

O número de elementos no CCD é, geralmente, diferente da dimensão em pixel da imagem capturada no *frame grabber*.



Então, as coordenadas de um ponto na imagem (X_{im}, Y_{im}) será diferente das coordenadas do mesmo ponto no CCD (X_{ccd}, Y_{ccd}) .

$$x_{im} = \frac{N}{n} \times x_{CCD} \qquad y_{im} = \frac{M}{m} \times y_{CCD}$$

Imagen digital

A imagen digital será dada por uma matriz $E_{M \times N}$, onde cada elemento $E(i, j)$ pode ser:

- Valor de intensidade luminosa de 0 a 255;
- Valor de profundidade de 0 a 255;
- Outro atributo, como temperatura, etc.

Fatos:

- A relação do valor $E(i, j)$ com o mundo físico depende do sensor e do processo de aquisição;
- Toda informação contida na imagem deve ser extraída da matriz $E_{M \times N}$

Imagen digital

A imagem de intensidade é determinada por uma série de parâmetros:

- **Parâmetros óticos:** Tipo de lente, foco, campo de visão, abertura angular;
- **Parâmetros fotométricos:** Intensidade e direção da luz, Propriedades das superfícies, reação dos sensores à luz;
- **Parâmetros geométricos:** Tipo de projeção, posição e orientação da câmera, distorção perspectiva;



Sistema ótico

Um único ponto da cena reflete luz em várias direções. Uma imagem tem *foco* quando a luz refletida por um ponto P na cena, atinge um único ponto p no plano da imagem. Isso pode ser obtido de duas formas:

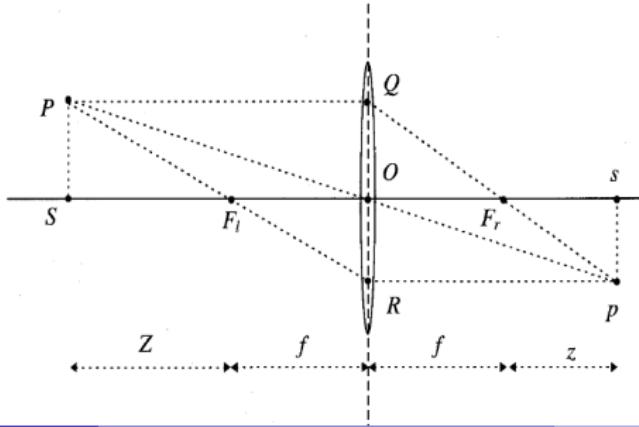
- reduzindo a abertura da câmera (pinhole);
- Usando um sistema ótico que direcione a p , na imagem, toda luz refletida por P , da cena.

Um sistema ótico, basicamente, produz uma *abertura pontual* usando uma abertura mais larga e uma exposição por menos tempo. Assim, o sistema ótico potencializa o aproveitamento da luz.

Sistema ótico

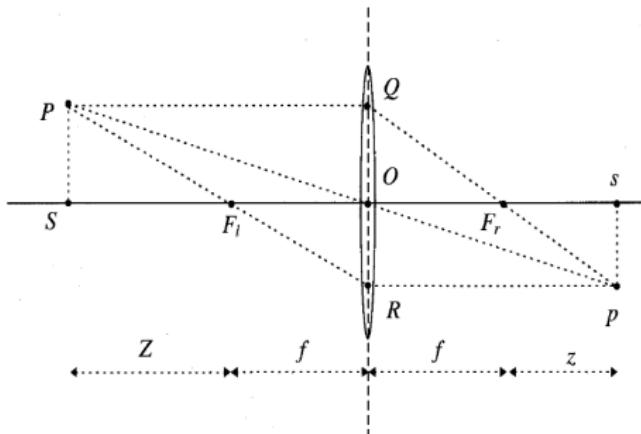
Propriedades das **lentes finas**:

- Raios que entram, parelelos ao eixo ótico de um lado, saem pelo foco, do outro lado;
- Raios que entram, passando pelo foco de um lado, saem paralelos ao eixo ótico, do outro lado.
- Raios que entram, passando pelo centro da lente, saem sem deflexão do outro lado.



Sistema ótico

Propriedades das **lentes finas**:



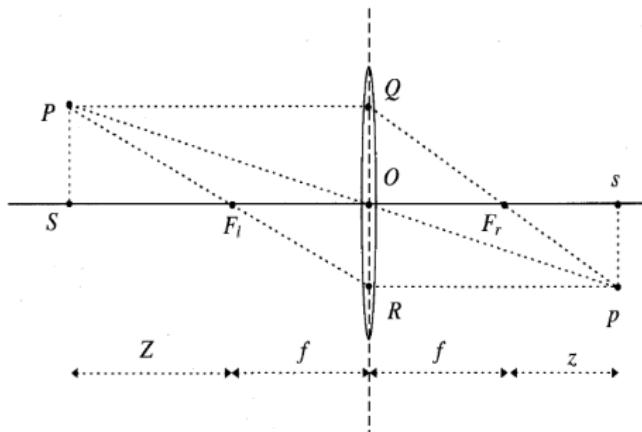
Observando que são semelhantes os triângulos $\triangle PF_lS$ e $\triangle ROF_l$ e também os triângulos $\triangle psF_r$ e $\triangle QOF_r$, temos que

$$\frac{Z}{f} = \frac{\overline{PS}}{\overline{OR}} \text{ e } \frac{f}{z} = \frac{\overline{QO}}{\overline{sp}} \quad \Rightarrow \quad \frac{Z}{f} = \frac{\overline{PS}}{\overline{OR}} \text{ e } \frac{f}{z} = \frac{\overline{PS}}{\overline{OR}} \quad \Rightarrow \quad \frac{Z}{f} = \frac{f}{z}$$

Então, $Z.z = f^2$

Sistema ótico

Propriedades das **lentes finas**:



De $Z.z = f^2$, obtemos

$$(Z + f - f).(z + f - f) = f^2$$

$$(Z + f).(z + f) - f.(z + f) - (Z + f).f + f^2 = f^2$$

$$(Z + f).(z + f) = f.(z + f) + (Z + f).f$$

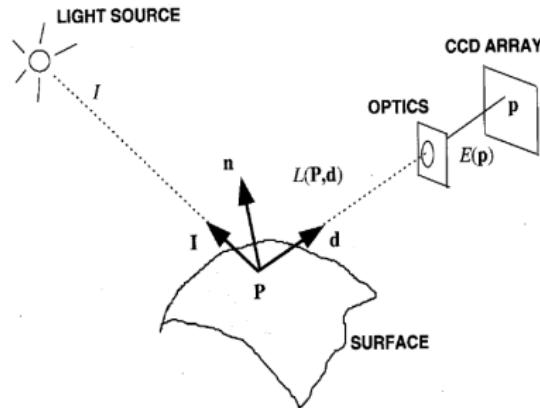
$$\frac{1}{f} = \frac{(z + f) + (Z + f)}{(Z + f).(z + f)} \Rightarrow$$

Equação Fundamental das lentes finas: $\frac{1}{f} = \frac{1}{Z + f} + \frac{1}{z + f}$

Sistema fotométrico

A radiometria trata da relação entre a quantidade de luz nos casos:

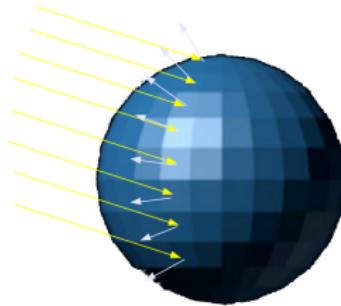
- Luz **emitida** pela fonte luminosa;
- Luz **refeletida** pela superfície;
- Luz **registrada** pelos sensores.



- **Radiância da cena:** Luz emitida pelo ponto P da cena em uma direção d ;
- **Irradiância da imagem:** Luz registrada pelo ponto p da imagem.

Sistema fotométrico

Modelo Lambertiano



No modelo Lambertiano, cada ponto reflete a luz igualmente em todas as direções. Materiais como papel e pintura fosca tem essa propriedade.

Neste caso, se a luz incide na direção do vetor \vec{I} e um ponto tem normal \vec{n} , a radiância L nesse ponto é proporcional ao produto escalar

$$L = \rho \cdot \vec{I} \cdot \vec{n}$$

Sistema fotométrico

Modelo Lambertiano, exemplo

```
function D=ilumina(imagem)

A=imread(imagem);
A=rgb2gray(A);
A=double(A);
[l,c]=size(A);

luz=[0,1,1]; luz=1/norm(luz,2)*luz;

Dif=zeros(l,c);

for i=4:l-4
    for j=4:c-4
        if (A(i,j)>1)
            u=[i+5,j,A(i+5,j)] - [i-5,j,A(i-5,j)];
            v=[i,j+5,A(i,j+5)] - [i,j-5,A(i,j-5)];
            n=cross(u,v);
            n=1/norm(n,2)*n;

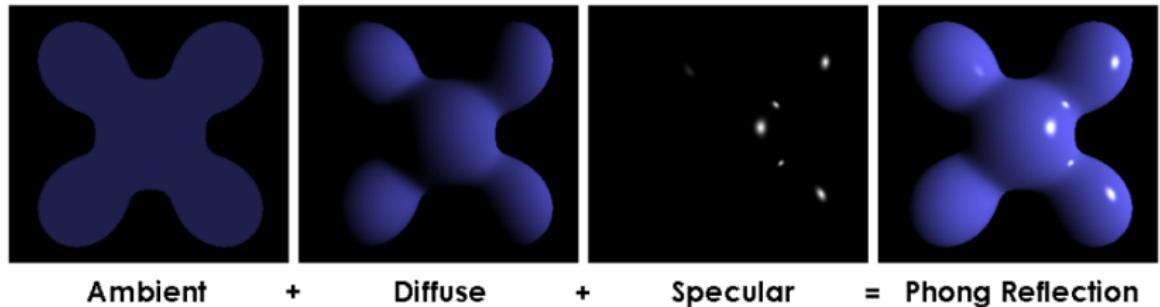
            Dif(i,j)=max(0,luz*n');

        end
    end
end

imshow(Dif);
```

Sistema fotométrico

Modelo de Phong



No modelo de Phong, mais completo, além da reflexão difusa da luz, é considerada uma luz ambiente e uma reflexão especular. Materiais como vidro e superfícies polidas apresentam especularidade.

$$L = L_a + L_d + L_s$$

Sistema fotométrico

Modelo Phong, exemplo

```
function D=iluminaphong(imagem)

A=imread(imagem);
A=double(rgb2gray(A));
[l,c]=size(A);

luz=[0,1,1]; luz=1/norm(luz,2)*luz;

o=[0,0,1];

D=zeros(l,c);
Amb=D; Dif=D; Esp=D;

for i=4:l-4
    for j=4:c-4
        if (A(i,j)>1)
            u=[i+5,j,A(i+5,j)] - [i-5,j,A(i-5,j)];
            v=[i,j+5,A(i,j+5)] - [i,j-5,A(i,j-5)];
            n=cross(u,v);
            n=1/norm(n,2)*n;

            l1=2*n*(n*luz')-luz;
            l1=1/norm(l1,2)*l1;

            Amb(i,j)=1;
            Dif(i,j)=max(0,luz*n');
            Esp(i,j)=max(0,(l1*o')11);

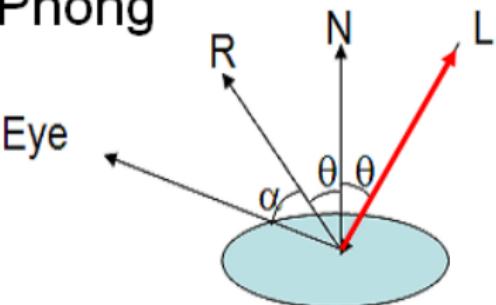
            D(i,j)= 0.2*Amb(i,j) + 0.4*Dif(i,j) + 0.4*Esp(i,j);

        end
    end

imshow([ Amb Dif ; Esp D ]);


```

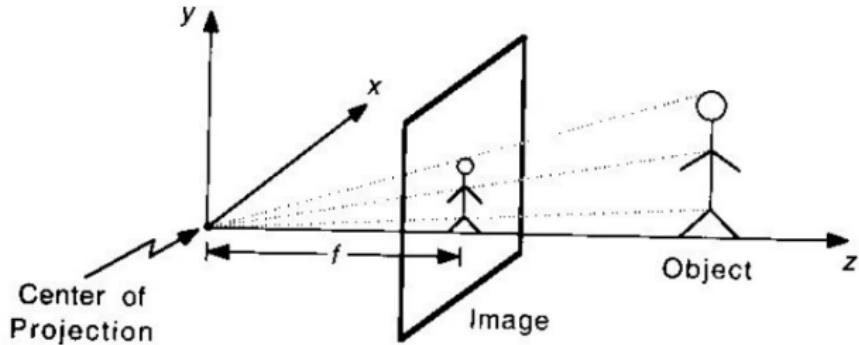
Phong



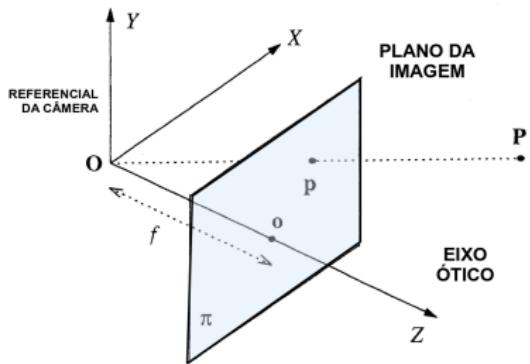
$$R = 2N(L \cdot N) - L$$

Modelo geométrico

Modelo geométrico de câmera sem imagem invertida



Modelo geométrico



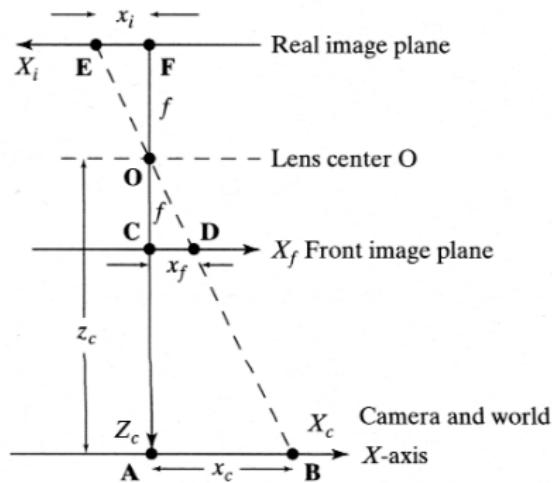
O modelo geométrico de câmera perspectiva consiste em um plano π que é o *plano da imagem* e um sistema de coordenadas com origem \mathbf{O} , chamado *centro de projeção*. A distância do ponto \mathbf{O} ao plano π , indicada por f , é a *distância focal*.

Nesse sistema, dado um ponto $P = [X, Y, Z]^T$ e sua imagem $p = [x, y, z]^t$ no plano π , são imediatas as **equações fundamentais**

$$x = f \cdot \frac{X}{Z} \quad \text{e} \quad y = f \cdot \frac{Y}{Z}$$

Modelo geométrico

Imagen 1D

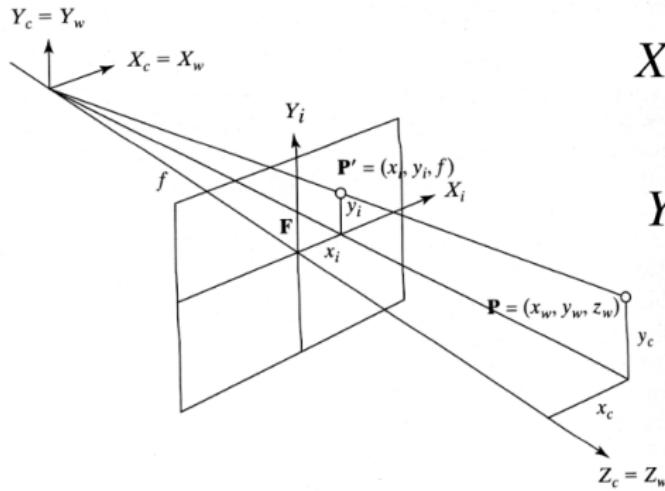


$$X_f / f = X_c / Z_c$$

$$X_f = f \frac{X_c}{Z_c}$$

Modelo geométrico

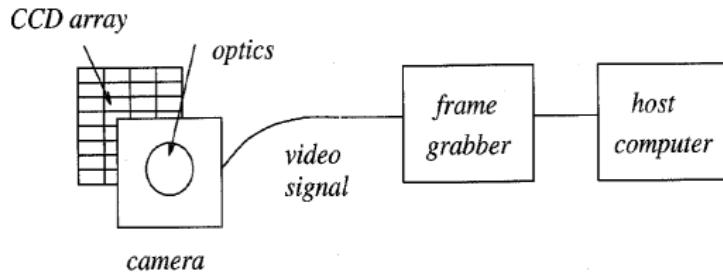
Imagen 2D



$$X_i = f \frac{X_c}{Z_c}$$

$$Y_i = f \frac{Y_c}{Z_c}$$

Formação da imagem



- A câmera recebe a luz, pela lente;
- O CCD, que é uma grade retangular de fotosensores, converte a luz em voltagem;
- A saída do CCD é um sinal elétrico obtido pela leitura dos sensores, linha por linha;
- O frame grabber digitaliza o sinal elétrico em uma matriz de inteiros E , de dimensão $M \times N$;
- A matriz E pode então ser processada numericamente em um computador.

Formação da imagem

Para propósito do curso de Visão Computacional, considera-se a imagem digitalizada E , com as propriedades:

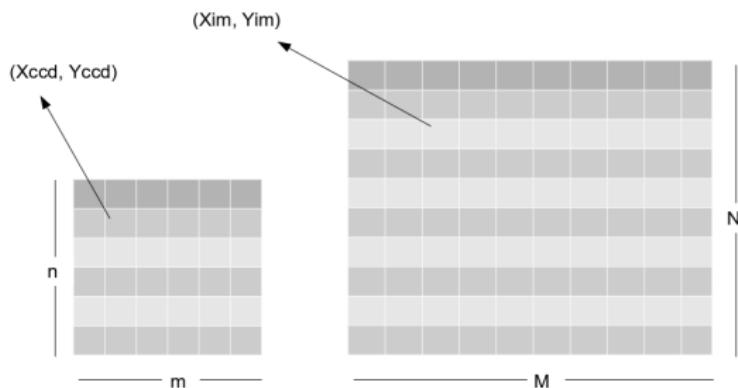
- E é uma matriz numérica com M linhas e N colunas;
- O elemento (i, j) será chamado **pixel** (*picture cell*);
- $E(i, j)$ indica a intensidade, ou brilho, no pixel (i, j) ;
- $E(i, j)$ será um inteiro, no intervalo $[0, 255]$.

Então, cada pixel assumir um dos 256 possíveis níveis de cinza sendo, por convenção, **0** preto e **255** branco.

Imagens coloridas podem ser representadas por três componentes monocromáticas (**R,G,B**), cada uma com 256 níveis.

Formação da imagem

O número de elementos no CCD é, geralmente, diferente da dimensão em pixel da imagem capturada no *frame grabber*.



Então, as coordenadas de um ponto na imagem (X_{im}, Y_{im}) será diferente das coordenadas do mesmo ponto no CCD (X_{ccd}, Y_{ccd}) .

$$x_{im} = \frac{N}{n} \times x_{CCD} \qquad y_{im} = \frac{M}{m} \times y_{CCD}$$

Formação da imagem

Assumindo que cada elemento do CCD mede $\mathbf{d} \times \mathbf{d}$ temos, pelo **Teorema da amostragem**, que a maior frequência espacial que pode ser capturada pela imagem é dada por

$$V_c = \frac{1}{2d}$$



Imagen digital

Propriedades físicas da matriz de fotosensores determinam outros parâmetros importantes na aquisição

- **Amostragem:** Natureza discreta da matriz de fotosensores;



- **Quantização:** Natureza discreta da escala de intensidade.

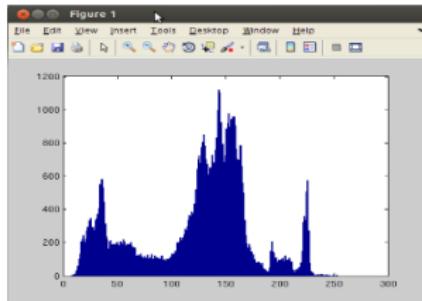


Histograma

Um histograma da imagem registra a frequência de ocorrência de cada nível de cinza.

```
A=imread('serro.bmp');
>> size(A)
>> imshow(A);
>> A=rgb2gray(A);
>> imshow(A);

>> myhist(A);
```

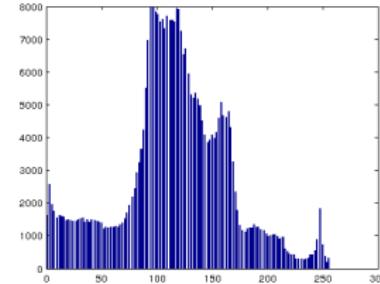
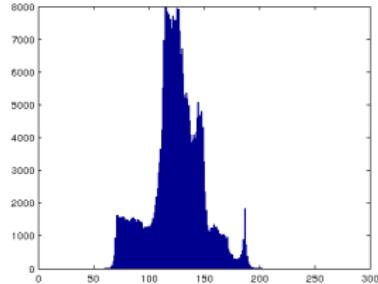


---myhist.m----

```
function h=myhist(A)
m=size(A,2);
n=size(A,1);
h=zeros(256,1);
for i=1:n
    for j=1:m
        k=A(i,j);
        h(k+1)=h(k+1)+1;
    end
end
bar(h)
```

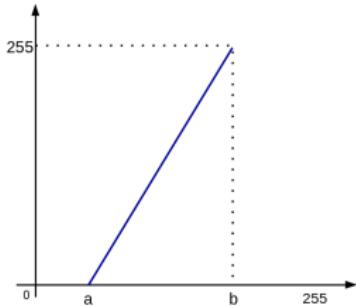
Equalização do histograma

A equalização do histograma redistribui os níveis de cinza.

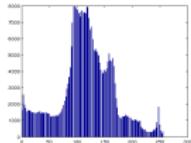
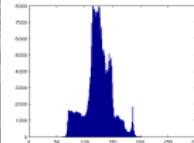


Equalização do histograma

```
function C=equaliza(B,a,b,t)
[m n]=size(B)
for i=1:m
    for j=1:n
        x=double(B(i,j));
        x=(255-0)/(b-a)*(x-a);
        x=abs(x/255)^t;
        C(i,j)=x*255;
    end
end
C=uint8(C);
```

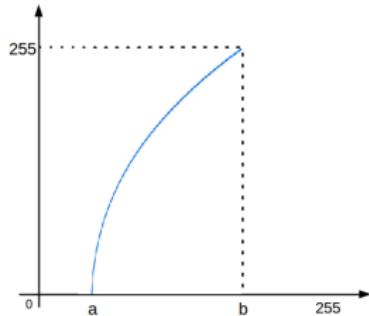


```
>> A imread('natal.png');
>> myhist(A);
>> a=min(min(A));
>> b=max(max(A));
>> C=equaliza(A,a,b,1);
>> myhist(C);
>> imshow([A C]);
```

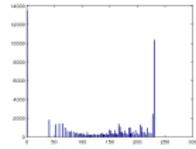
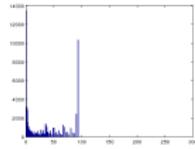


Equalização do histograma

```
function C=equaliza(B,a,b,t)
[m n]=size(B)
for i=1:m
    for j=1:n
        x=double(B(i,j));
        x=(255-a)/(b-a)*(x-a);
        x=abs(x/255)^t;
        C(i,j)=x*255;
    end
end
C=uint8(C);
```



```
>> A=imread('vw.png');
>> myhist(A);
>> a=min(min(A));
>> b=max(max(A));
>> C=equaliza(A,a,b,1/2);
>> myhist(C);
>> imshow([A C]);
```



Ruído na formação da imagem

Dado um conjunto de imagens E_0, E_1, \dots, E_{n-1} de tamanho $N \times N$, temos, para cada pixel (i, j) , um valor médio $\overline{E}(i, j)$ e uma estimativa do ruído $\sigma(i, j)$:

- $\overline{E}(i, j) = \frac{1}{n} \times \sum_{k=0}^{n-1} E_k(i, j)$
- $\sigma(i, j) = \sqrt{\frac{1}{n-1} \times \sum_{k=0}^{n-1} (\overline{E}(i, j) - E_k(i, j))^2}$



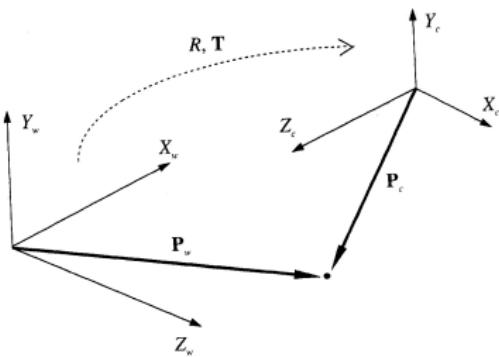
Parâmetros da câmera

A utilidade da *Visão Computacional* está em associar **pixels** da imagem com **coordenadas 3D** de pontos da cena.

Essa associação depende do conhecimento de um conjunto de parâmetros **intrínsecos** e **extrínsecos**.

- Os parâmetros **Extrínsecos** definem a localização e orientação da câmera com respeito a um sistema de coordenadas conhecido do mundo;
- Os parâmetros **Intrínsecos** definem a transformação de um ponto no sistema de coordenadas da câmera para coordenadas de pixel numa imagem.

Parâmetros da câmera: Extrínsecos



- Matriz de rotação $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$;
- Vetor de translação $T = [T_x, T_y, T_z]$.

R e T , chamados **parâmetros extrínsecos**, são usados para estabelecer a relação

$$P_c = R(P_w - T)$$

Parâmetros da câmera: Intrínsecos

Os parâmetros *intrínsecos* são:

- A distância focal f ;
- O tamanho do pixel, em milímetros, s_x e s_y
- O centro da imagem (o_x , o_y)

As coordenadas, em pixels $p_{im} = [x_{im}, y_{im}]$, são obtidas das coordenadas $P_c = [X_c, Y_c, Z_c]$ no sistema de referência da câmera fazendo

$$x_{im} = o_x - \frac{f}{s_x} \frac{X_c}{Z_c} \quad \text{e} \quad y_{im} = o_y - \frac{f}{s_y} \frac{Y_c}{Z_c}$$

Na forma matricial, os parâmetros intrínsecos definem a matriz M_{int} , dada por

$$M_{int} = \begin{bmatrix} \frac{-f}{s_x} & 0 & o_x \\ 0 & \frac{-f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix}.$$

Parâmetros da câmera

Plugando os parâmetros extrínsecos e intrínsecos, obtemos uma **versão linear** para as equações de projeção perspectiva

$$\text{De } x_{im} = o_x - \frac{f}{s_x} \frac{X_c}{Z_c}, \quad \text{obtemos} \quad -(x_{im} - o_x).s_x = f \frac{X_c}{Z_c}$$

$$\text{De } y_{im} = o_y - \frac{f}{s_y} \frac{Y_c}{Z_c}, \quad \text{obtemos} \quad -(y_{im} - o_y).s_y = f \frac{Y_c}{Z_c}$$

Fazendo $X_c = R_1^T(P_w - T)$, $Y_c = R_2^T(P_w - T)$ e $Z_c = R_3^T(P_w - T)$, temos

$$-(x_{im} - o_x).s_x = f \frac{R_1^T(P_w - T)}{R_3^T(P_w - T)}$$

$$-(y_{im} - o_y).s_y = f \frac{R_2^T(P_w - T)}{R_3^T(P_w - T)}$$

Parâmetros da câmera

De $-(x_{im} - o_x) \cdot s_x = f \frac{R_1^T(P_w - T)}{R_3^T(P_w - T)}$ e $-(y_{im} - o_y) \cdot s_y = f \frac{R_2^T(P_w - T)}{R_3^T(P_w - T)}$

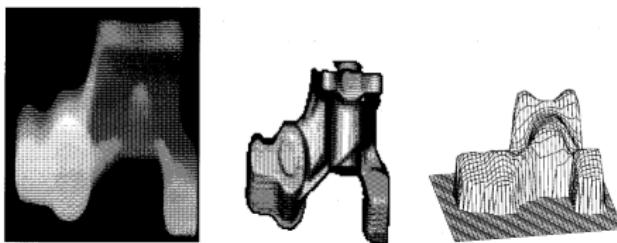
E tomando $M_{ext} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & -R_1^T T \\ r_{21} & r_{22} & r_{23} & -R_2^T T \\ r_{31} & r_{32} & r_{33} & -R_3^T T \end{bmatrix}$ e $M_{int} = \begin{bmatrix} \frac{-f}{s_x} & 0 & o_x \\ 0 & \frac{-f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix}$.

Podemos escrever $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = M_{int} \times M_{ext} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$.

Donde, finalmente, $x_{im} = \frac{x_1}{x_3}$ e $y_{im} = \frac{x_2}{x_3}$

Imagen de profundidade

Uma imagem de profundidade é uma imagem digital E , onde cada pixel $E(i, j)$ expressa a profundidade de um ponto na cena em relação a um sistema de referência conhecido.



Imagens de profundidade podem ser representadas como:

- Matriz E , onde $r_{ij} = E(i, j)$ é a profundidade no pixel (i, j) ;
- Lista de coordenadas 3D (nuvem de pontos).

Imagen de profundidade

Naturalmente, uma imagem de profundidade pode ser visualizada como uma imagem de intensidade.



Um sensor de profundidade pode ser:

- **Sensor ativo:** Projeta alguma energia (padrão de luz, sonar, etc) e estima a profundidade de acordo com o sinal lido de volta.
- **Sensor passivo:** Usa imagens de intensidade comuns (geralmente mais de uma imagem) para fazer a reconstrução.

Imagen de profundidade

Sensores passivos geralmente combinam mais de uma imagem de intensidade, obtidas em um sistema calibrado usando *geometria epipolar*.

Esse processo de estimativa da profundidade a partir da intensidade é chamado **reconstrução 3D**.

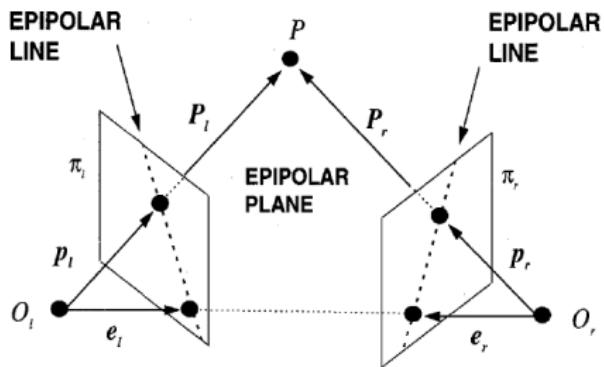


Imagen de profundidade

Sensores ativos exploram princípios físicos como:

- **Radar ou sonar:** Uma onda eletromagnética ou sonora é emitida e o tempo que leva para atingir uma superfície e retornar permite estimar distâncias.
- **Luz estruturada:** Um padrão de linhas sobrepostas ou grade é projetado na superfície e a deformação do padrão, capturada em uma imagem, é processada para estimar a geometria da cena.
- **Foco ativo:** Uma lente motorizada varia o foco continuamente e um modelo, baseado na equação fundamental das lentes finas, associa o valor do foco com a profundidade.

Imagen de profundidade

Sensores ativos mais comuns usam **Triangulação Ativa**:

A triangulação ativa usa um projetor de luz a uma distância **b** do centro de projeção da câmera.

O projetor emite um plano de luz formando um ângulo controlado θ . A interseção da luz com a superfície define uma linha que é capturada pela câmera.

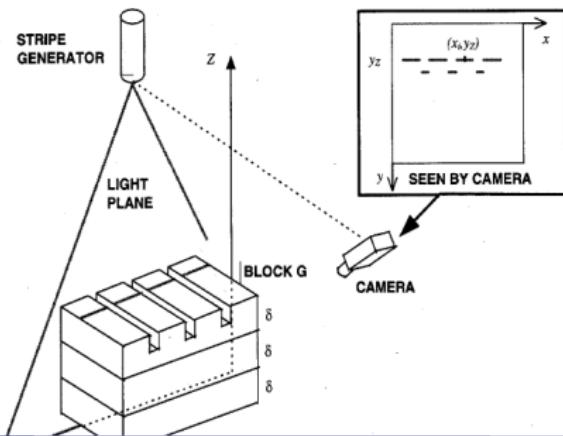
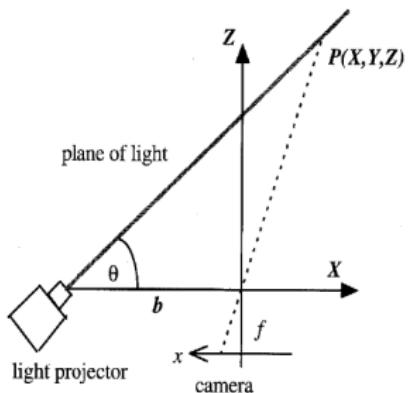


Imagen de profundidade



As coordenadas 3D de um ponto $P = [X, Y, Z]$ na linha projetada são obtidas a partir do pixel (x, y) , foco f , e θ , como:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \frac{b}{f \cdot \cot(\theta) - x} \begin{pmatrix} x \\ y \\ f \end{pmatrix}$$

Ruído e filtragem

Em visão computacional, **ruído** é qualquer entidade, na imagem ou resultado intermediário, que não é interessante ao propósito do processamento. Por exemplo:

- No processamento de imagem, como detecção de arestas, ruído pode ser qualquer flutuação no valor do pixel, introduzido pelo sistema de aquisição;
- Quando um algoritmo recebe como entrada o resultado de alguma computação numérica, ruído pode ser o erro introduzido por arredondamento ou limitação da precisão numérica;
- Quando identificamos linhas de contorno em objetos numa imagem, ruído pode ser algum contorno detectado na imagem que não tem relação com objetos da cena.

Ruído e filtragem

Assume-se que o ruído em imagens é *aditivo* e *randômico*, ou seja, um sinal espúrio, randômico $n(i, j)$, adicionado ao real valor do pixel $I(i, j)$, que resulta na imagem ruidosa $\hat{I}(i, j)$.

$$\hat{I}(i, j) = I(i, j) + n(i, j)$$

A quantidade de ruído na imagem pode ser estimada por σ_n , desvio padrão do sinal randômico $n(i, j)$. A **razão-sinal-ruído** indica o quanto o ruído é significativo em relação ao sinal, e é obtido por

$$SNR = \frac{\sigma_s}{\sigma_n}$$

Ruído e filtragem

Ruído gaussiano:

Na falta de maiores informações, assume-se que $n(i, j)$ é modelado por uma distribuição *gaussiana* de média zero.

Para cada pixel (i, j) , o ruído $n(i, j)$ é um valor aleatório, distribuído simetricamente em relação ao suposto valor real $I(i, j)$ do pixel.



```
function E=noiseg(E,k)
[l,c]=size(E);
for i=1:l
    for j=1:c
        E(i,j)=E(i,j)+normrnd(0,k);
    end
end
```



Ruído e filtragem

Ruído gaussiano:



```
function E=noiseg(E,k)
[l,c]=size(E);
for i=1:l
    for j=1:c
        E(i,j)=E(i,j)+normrnd(0,k);
    end
end
```



```
» A=imread('serro.bmp');
» A=rgb2gray(A);
» imshow(A);
» B=noiseg(A,30);
» imshow(B);
```

Ruído e filtragem

Ruído impulsivo:

O ruído *impulsivo*, conhecido com *sal e pimenta*, ocorre normalmente por erro em pixels randômicos, tornando seu valor muito diferente do real de forma a se destacar dos pixels vizinhos.

Para cada pixel (i, j) , o ruído impulsivo $n(i, j)$ é modelado como:

$$n(i, j) = \begin{cases} I(i, j) & \text{se } x \geq t \\ v_{min} + y(v_{max} - v_{min}) & \text{se } x < t \end{cases}$$

- $I(i, j)$ é o valor real no pixel (i, j)
- $x, y \in [0, 1]$ são duas variáveis randômicas
- t é um parâmetro que controla o quanto da imagem é corrompida

Ruído e filtragem

Ruído impulsivo:

$$n(i,j) = \begin{cases} I(i,j) & \text{se } x \geq t \\ v_{min} + y(v_{max} - v_{min}) & \text{se } x < t \end{cases}$$



```
function E=noisei(E,k,vmin,vmax)
[l,c]=size(E);

for i=1:l
    for j=1:c
        x=rand;
        y=rand;
        if (x<k)
            E(i,j)=vmin + y*(vmax - vmin);
        end
    end
end
```



```
» A=imread('serro.bmp');
» A=rgb2gray(A);
» imshow(A);
» B=noisei(A,0.1,0,255);
» imshow(B);
```

Ruído e filtragem

Filtro linear:

Atenuar ou, se possível, remover ruído é um importante tópico em processamento de imagens.

A técnica mais comum para *suavizar* o ruído é a **filtragem linear**, que consiste em uma *convolução* da imagem com uma matriz constante, chamada **máscara**, ou *kernel*.

1	11	4	5	7	2	0	17	8
6	3	19	20	11	10	7	1	1
0	1	6	2	5	0	7	5	2
7	9	7	6	4	11	7	8	9
7	5	13	7	5	8	7	5	42
7	9	7	2	6	8	9	5	12
2	6	7	4	2	1	4	5	7
3	2	1	6	15	0	9	6	3
2	1	5	7	9	10	9	5	3

6	4	11
7	5	8
2	6	8

X

1	2	1
2	3	2
1	2	1

1	11	4	5	7	2	0	17	8
6	3	19	20	11	10	7	1	1
0	1	6	2	5	0	7	5	2
7	9	7	6	4	11	7	8	9
7	5	13	7	6	8	7	5	42
7	9	7	2	6	8	9	5	12
2	6	7	4	2	1	4	5	7
3	2	1	6	15	0	9	6	3
2	1	5	7	9	10	9	5	3

$$(6.1 + 4.2 + 11.1 + 7.2 + 5.3 + 8.2 + 2.1 + 6.2 + 8.1) / 15 = 6$$

Ruído e filtragem

Filtro linear:

1	11	4	5	7	2	0	17	8
6	3	19	20	11	10	7	1	1
0	1	6	2	5	0	7	5	2
7	9	7	6	4	11	7	8	9
7	5	13	7	5	8	7	5	42
7	9	7	2	6	8	9	5	12
2	6	7	4	2	1	4	5	7
3	2	1	6	15	0	9	6	3
2	1	5	7	9	10	9	5	3

6	4	11
7	5	8
2	6	8

X

1	2	1
2	3	2
1	2	1

1	11	4	5	7	2	0	17	8
6	3	19	20	11	10	7	1	1
0	1	6	2	5	0	7	5	2
7	9	7	6	4	11	7	8	9
7	5	13	7	6	8	7	5	42
7	9	7	2	6	8	9	5	12
2	6	7	4	2	1	4	5	7
3	2	1	6	15	0	9	6	3
2	1	5	7	9	10	9	5	3

$$(6.1 + 4.2 + 11.1 + 7.2 + 5.3 + 8.2 + 2.1 + 6.2 + 8.1) / 15 = 6$$

Dada uma imagem I , $N \times M$, um número ímpar m menor que M, N , e uma máscara A , $m \times m$, a versão filtrada I_A de I é dada por

$$I_A(i, j) = I * A = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} [A(h, k) \times I(i - h, j - k)]$$

Ruído e filtragem

Filtro linear:

```
function IA=filtrolinear(I,A)

[l,c]=size(I);
n=size(A,1);
m=floor(n/2);

IA=zeros(l,c);
I=double(I);

for i=(m+1):(l-m)
    for j=(m+1):(c-m)
        for h=-m:m
            for k=-m:m
                IA(i,j)=IA(i,j)+A(h+m+1,k+m+1)*I(i-h,j-k);
            end
        end
    end
end

if (sum(sum(A))>0)
    IA=IA/sum(sum(A));
end

IA=uint8(IA);
```

Ruído e filtragem

Filtro linear:

Algumas máscaras para filtro linear:

$$\text{Filtro da média: } A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \frac{1}{9}$$

$$\text{Filtro da média: } A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \frac{1}{25}$$

$$\text{Filtro da Gaussiano: } A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16}$$

Ruído e filtragem

O termo **filtragem** vem do fato de que, no domínio da frequência, a convolução tem o efeito de remover, ou filtrar, sinais em determinadas frequências.

Os filtros, de acordo com o espectro de frequência que preservam, são classificados em:

- Filtro **passa-baixa**: Filtram altas frequências
- Filtro **passa-alta**: Filtram baixas frequências
- Filtro **passa-faixa**: Filtram frequências em determinada faixa

A transformada discreta de **fourier** em 2D permite analizar os filtros e imagens no domínio da frequência de forma a compreender o processo de filtragem.

Ruído e filtragem

Conforme o **Teorema da convolução**, a convolução de I com A é simplesmente o produto das transformadas $\mathcal{F}(I)$ e $\mathcal{F}(A)$.

$$I * A = \mathcal{F}^{-1}(\mathcal{F}(I)\mathcal{F}(A)).$$

onde \mathcal{F} indica a **transformada de Fourier** e \mathcal{F}^{-1} a transformada inversa.

No caso de imagens, uma versão discreta, chamada **DFT**, ou *Discrete fourier transform* é utilizada.

O algoritmo mais comum para essa transformação é o **FFT**, ou *fast fourier transform*

Ruído e filtragem

A transformada discreta de Fourier, e sua inversa, para um sinal I , são definidas como:

$$\mathcal{F}(I(x)) = F(w) = \frac{1}{\sqrt{n}} \times \sum_{x=0}^{n-1} I(x) \times e^{-\frac{2i\pi wx}{n}}.$$

$$\mathcal{F}^{-1}(F(w)) = I(x) = \frac{1}{\sqrt{n}} \times \sum_{w=0}^{n-1} F(w) \times e^{\frac{2i\pi wx}{n}}.$$

Ou Equivalentemente,

$$\mathcal{F}(I(x)) = F(w) = \frac{1}{\sqrt{n}} \times \sum_{x=0}^{n-1} I(x) \times (\cos(\frac{-2\pi wx}{n}) - i \cdot \sin(\frac{-2\pi wx}{n})).$$

$$\mathcal{F}^{-1}(F(w)) = I(x) = \frac{1}{\sqrt{n}} \times \sum_{w=0}^{n-1} F(w) \times (\cos(\frac{2\pi wx}{n}) - i \cdot \sin(\frac{2\pi wx}{n})).$$

Ruído e filtragem

Em Matlab, podemos escrever o código:

```
function F = DFT1D(I,s)
n = length(I);
F = zeros(1,n);
for w=0:n-1
    for x=0:n-1
        F(w+1)=F(w+1) + 1/sqrt(n)*I(x+1)*(cos(-s*2*pi*w*x/n) - i*sin(-s*2*pi*w*x/n));
    end
end
end
```

Ruído e filtragem

A transformada discreta de Fourier, e sua inversa, para uma imagem I , $m \times n$, são definidas como:

$$\mathcal{F}(I(x, y)) = F(w, z) = \frac{1}{\sqrt{mn}} \times \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} I(x, y) \times e^{-2i\pi(\frac{wx}{n} + \frac{yz}{m})}$$

$$\mathcal{F}^{-1}(F(w, z)) = I(x, y) = \frac{1}{\sqrt{mn}} \times \sum_{w=0}^{n-1} \sum_{z=0}^{m-1} F(w, z) \times e^{2i\pi(\frac{wx}{n} + \frac{yz}{m})}$$

Ruído e filtragem

Em Matlab, podemos escrever o código:

```
function F = DFT2D(I,s)
[m n] = size(I);
F=double(zeros(m,n));
for w=0:(m-1)
    for z=0:(n-1)
        for x=0:(m-1)
            for y=0:(n-1)
                F(w+1,z+1) = F(w+1,z+1) + 1/sqrt(m*n)*I(x+1,y+1) * exp(-s*2i*pi*( w*x/n + z*y/m ));
```

Ruído e filtragem

O processamento no **domínio da frequência** é realizado em 3 passos:

- A imagem, no domínio espacial \mathbf{A} , é transformada para $F = \mathcal{F}(A)$, no domínio da frequência;
- Alguma operação, ou transformação, $B = T(F)$ é realizado no domínio da frequência;
- A imagem B , modificada no domínio da frequência, é transformada para o domínio espacial, $C = \mathcal{F}^{-1}(B)$ pela transformada inversa de Fourier

Em Matlab:

- $F = fft2(A)$, transformada rápida de fourier em 2D;
- $B = T(F)$, qualquer modificação, geralmente suprimindo frequências altas, baixas ou faixas;
- $C = ifft2(B)$.

Ruído e filtragem

```
function F=freqfilter(A,minf,maxf)
[l,c]=size(A);

F=fft2(A); % Transformada de fourier de A
F=fftshift(F);

for i=1:l
    for j=1:c
        r=sqrt((l/2-i)^2+(c/2-j)^2);
        if (r>minf & r<maxf)
            F(i,j)=0; % Elimina frequências
        end
    end
end

IF=ifft2(F); % Transformada inversa

S=(log(abs(F)+1));
S=S/max(max(S))*255; % Mostra o espectro
IF=abs(IF); % Mostra a transformada
imshow([A S IF]);
```

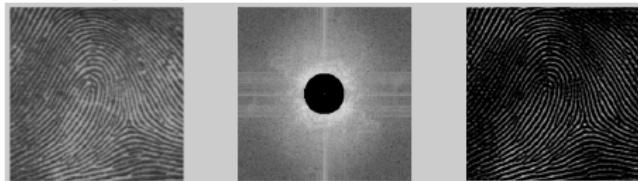
Ruído e filtragem

```
» I=imread('digital.png');  
» I=rgb2gray(I);
```

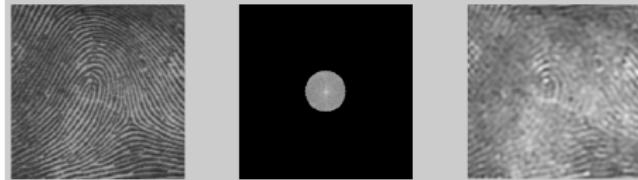
```
» F=freqfilter(I,0,0);
```



```
» F=freqfilter(I,0,30);
```



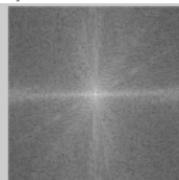
```
» F=freqfilter(I,30,1000);
```



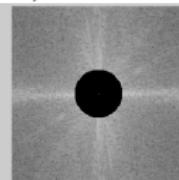
Ruído e filtragem

```
» I=imread('serro.bmp');  
» I=rgb2gray(I);
```

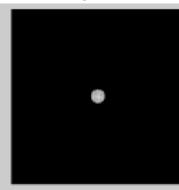
```
» F=freqfilter(I,0,0);
```



```
» F=freqfilter(I,0,10);
```



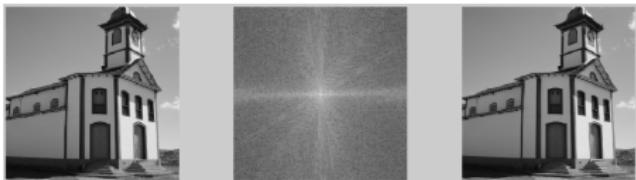
```
» F=freqfilter(I,10,1000);
```



Ruído e filtragem

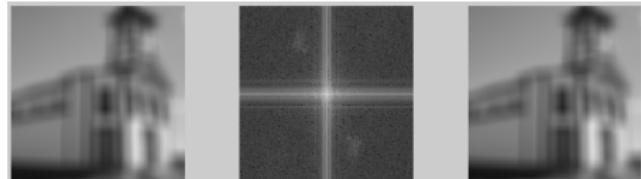
O filtro da média atua, no domínio da frequência, como filtro **passa-baixa**

```
» I=imread('serro.bmp');  
» I=rgb2gray(I);
```



```
» F=freqfilter(I,0,0);
```

```
» A=[1 1 1;1 1 1;1 1 1]  
» B=filtrolinear(I,A);
```



```
» F=freqfilter(B,0,0);
```

Ruído e filtragem

O filtro **Gaussiano**, no domínio da frequência, continua uma Gaussiana. Essa propriedade faz desse filtro uma melhor opção que o filtro da média para suavização.

Outra vantagem do filtro gaussiano, é em termos de desempenho.

Esse filtro tem a propriedade de separabilidade da máscara. Isto implica que, fazer a convolução com uma máscara $2D$ é equivalente convolução $1D$ em todas as linhas, seguido da convolução $1D$ em todas as colunas.

$$I * G = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} [G(h, k) \times I(i - h, j - k)]$$

$$I * G = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} \left[e^{-\frac{h^2+k^2}{2\sigma^2}} \times I(i - h, j - k) \right]$$

$$I * G = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{h^2}{2\sigma^2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{k^2}{2\sigma^2}} \times I(i - h, j - k)$$

Ruído e filtragem

Um **kernel Gaussiano** G é obtido a partir do tamanho da máscara n ímpar e do valor do desvio padrão s .

$$G(h, k) = e^{-\frac{h^2+k^2}{2\sigma^2}}$$

Em Matlab:

```
function G=gaussian(n,s);
m=floor(n/2);

for h=-m:m
    for k=-m:m
        G(h+m+1,k+m+1)=exp(-( (h)^2 + (k)^2 )/(2*s^2));
    end
end

G=G/min(min(G));
G=floor(G);

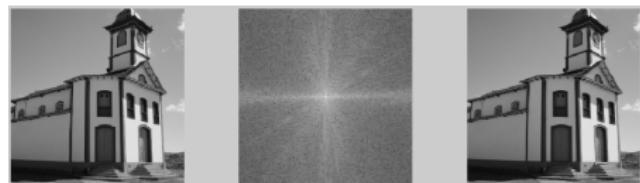
» G=gaussian(5,1);
» G=gaussian(15,2);
» surf(G)

» B=filrolinear(I,G);
```

Ruído e filtragem

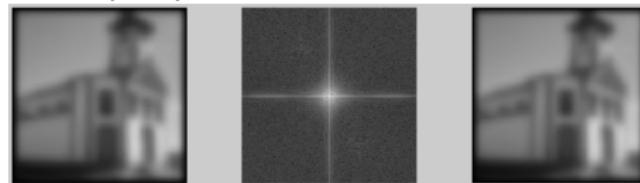
O filtro **Gaussiano** é uma melhor opção que o filtro da média para suavização.

```
» B=imread('serro.bmp');  
» B=rgb2gray(B);
```



```
» F=freqfilter(B,0,0);
```

```
» A=gaussian(5,1);  
» for i=1:100; B=filtrolinear(B,A); end
```



```
» F=freqfilter(B,0,0);
```

Ruído e filtragem

O ruído gaussiano é atenuado com o filtro linear gaussiano. Por exemplo:

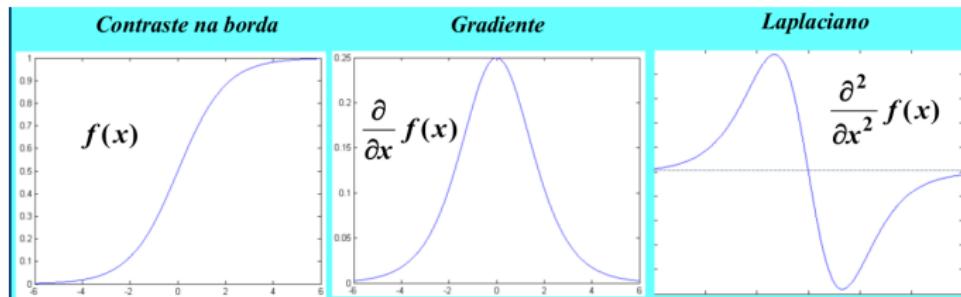


```
» A=imread('serro.bmp');
» A=rgb2gray(A);
» B=noiseg(A,60);
» G=gaussian(5,1);
» C=filtrolinear(B,G);
» imshow([A, B, C]);
```

Ruído e filtragem

O filtro **Laplaciano** é um operador escalar baseado na derivada de 2^a ordem.

$$\nabla^2 f(x, y) = \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y)$$



$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \Rightarrow \nabla = \begin{bmatrix} (b-e) & (b-e) \\ (e-d) & (f-e) \\ (e-h) & (e-h) \end{bmatrix}$$

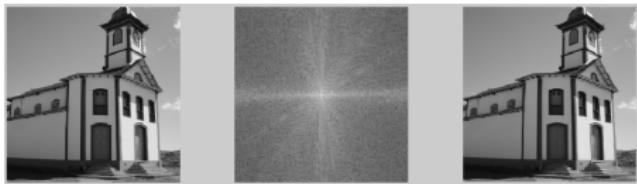
$$\nabla^2 = [(f-e) - (e-d)] + [(b-e) - (e-h)] = f + d + b + h - 4e$$

$$\text{Máscara } L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Ruído e filtragem

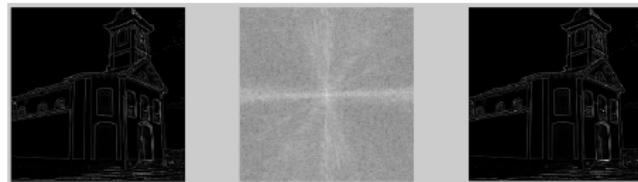
O filtro laplaciano atua, no domínio da frequência, como filtro **passa-alta**

```
» I=imread('serro.bmp');  
» I=rgb2gray(I);
```



```
» F=freqfilter(I,0,0);
```

```
» A=[0 1 0;1 -4 1;0 1 0]  
» B=filtrolinear(I,A);
```



```
» F=freqfilter(B,0,0);
```

Ruído e filtragem

Usando o Teorema da Convolução, o **Filtro Laplaciano** pode ser obtido como:

```
» I=imread('serro.bmp');
» I=rgb2gray(I);
» L=[0 1 0;1 -4 1;0 1 0];

» FI=fft2(I);
» FL=fft2(L,256,256);
» B=ifft2(FI.*FL);
» imshow([I B]);
```

Que, equivale a:

```
» C=filrolinear(I,L);
» imshow(C);
```

Ruído e filtragem

Usando o Teorema da Convolução, o **Filtro Gaussiano** pode ser obtido como:

```
» I=imread('serro.bmp');
» I=rgb2gray(I);
» G=[1 2 1;2 4 2;1 2 1];
» G=G/sum(sum(G));

» FI=fft2(I);
» FG=fft2(G,256,256);
» B=ifft2(FI.*FG);
» imshow([I B]);
```

Que, equivale a:

```
» C=filrolinear(I,G);
» imshow(C);
```

Ruído e filtragem

Um filtro não linear

O filtro da mediana é um exemplo de filtro que não pode ser modelado com uma máscara, não é linear.

```
function B=mediana(A)
[m,n]=size(A);
B=A;
for i=2:m-1
    for j=2:n-1
        x=median(median(A(i-1:i+1,j-1:j+1)));
        B(i,j)=x;
    end
end
B=uint8(B);
```

1	11	4	5	7	2	0	17	8
6	3	19	20	11	10	7	1	1
0	1	6	2	5	0	7	5	2
7	9	7	6	4	11	7	8	9
7	5	13	7	5	8	7	5	42
7	9	7	2	6	8	9	5	12
2	6	7	4	2	1	4	5	7
3	2	1	6	15	0	9	6	3
2	1	5	7	9	10	9	5	3

6	4	11
7	5	8
2	6	8

1	11	4	5	7	2	0	17	8
6	3	19	20	11	10	7	1	1
0	1	6	2	5	0	7	5	2
7	9	7	6	4	11	7	8	9
7	5	13	7	6	8	7	5	42
7	9	7	2	6	8	9	5	12
2	6	7	4	2	1	4	5	7
3	2	1	6	15	0	9	6	3
2	1	5	7	9	10	9	5	3

$$\text{Mediana}(2 - 4 - 5 - 6 - 6 - 7 - 8 - 8 - 11) = 6$$

Ruído e filtragem

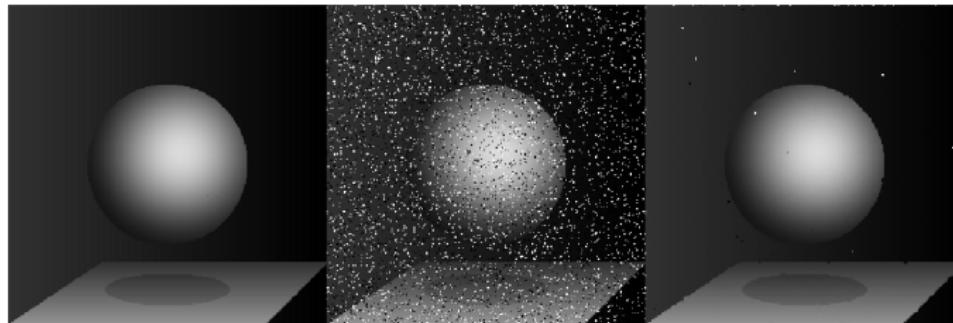
O ruído impulsivo é atenuado com o filtro não linear, da mediana. Por exemplo:



```
» A=imread('serro.bmp');
» A=rgb2gray(A);
» B=noisei(A,0.1,0,255);
» C=mediana(B);
» imshow([A, B, C]);
```

Ruído e filtragem

O ruído impulsivo é atenuado com o filtro não linear, da mediana. Por exemplo:



```
» A=imread('esfera.png');  
» A=rgb2gray(A);  
» B=noisei(A,0.1,0,255);  
» C=mediana(B);  
» imshow([A, B, C]);
```

Features em imagens

O tema em inglês **image feature** pode ser traduzido como *características da imagem ou ponto característico na imagem*. Usaremos sem tradução a palavra feature. Uma feature pode ser:

- **Global:** Uma propriedade global da imagem com intensidade média, quantidade de níveis de cinza, etc.
- **Local:** Uma parte da imagem com alguma propriedade especial definindo um ponto, uma linha, um círculo, etc.

Detectar features em imagens é parte fundamental da maioria das operações em Visão Computacional e se dá nos passos:

- **Detectar** pontos de interesse com características relevantes na imagem;
- **Agrupar** os pontos e classificar em linhas, cantos, círculos, etc.

Detecção de arestas

Pontos de aresta, ou simplesmente **arestas** são pixels onde a imagem tem uma variação brusca de intensidade.

A detecção é feita em três passos:

- **Suavização de ruído:** Remove a maior quantidade possível de ruído, sem destruir as arestas verdadeiras;
- **Destaque de arestas:** Executa um filtro que responde a arestas, ou seja, um filtro com resposta alta nas arestas e baixa em regiões suaves;
- **Localização:** Decide que valor retornado pelo filtro será considerado aresta e qual será descartado como ruído.

Detecção de arestas

Diversos algoritmos são propostos para detectar arestas:

- **Canny**: Método sofisticado em 3 passos, obtém linhas com expressura de pixel;
- **Roberts**: Baseado em filtro linear;
- **Sobel**: Baseado no gradiente obtido por filtro linear;

Detecção de arestas

Exemplo: Algoritmo SOBEL para detecção de arestas na imagem A:

- 1 Recebe A, aplica filtro gaussiano para suavização, obtendo B;
- 2 Filtra B com as máscaras de detecção X e Y, obtendo DX e DY.

$$X = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \text{ e } Y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

- 3 Estima a magnitude do gradiente $G(i, j)$ em cada pixel (i, j) , fazendo

$$G(i, j) = \sqrt{DX(i, j)^2 + DY(i, j)^2}$$

- 4 Marca como aresta todos os pixels (i, j) tais que $G(i, j) > \tau$

Detecção de arestas

Implementação do algoritmo SOBEL

```
function S=sobel(image,t)

A=imread(image);
A=rgb2gray(A);

[l,c]=size(A);

G=gaussian(5,1);
B=filtrolinear(A,G);

X=[-1 -2 -1;  0 0 0;  1 2 1];
Y=[-1  0  1; -2 0 2; -1 0 1];

DX=double(filtrolinear(B,X));
DY=double(filtrolinear(B,Y));

S=zeros(l,c);
for i=5:l-5
    for j=5:c-5
        G(i,j)=sqrt(DX(i,j)^2 + DY(i,j)^2);
        if (G(i,j)>t)
            S(i,j)=255;
        end
    end
end

imshow([A, S]);
```

Detecção de arestas

Exemplo de aplicação do filtro SOBEL



```
» S=sobel('serro2.bmp',50);
```

Detecção de arestas

Exemplo de aplicação do filtro SOBEL



```
» S=sobel('curvas.png',50);
```

Detecção de cantos

Cantos, ou quinas, são pixels onde a imagem tem uma variação brusca de intensidade em mais de uma direção.

A detecção é feita usando análise de componentes principais (PCA) do gradiente em uma vizinhança.

- ① Computa o gradiente $[D_x, D_y]$ para todo ponto da imagem
- ② Para cada ponto p da imagem:
 - ① Obtém a matriz C de covariância do gradiente em uma vizinhança de p
 - ② Encontra λ_2 , o menor dos autovalores de C
 - ③ Se $\lambda_2 > \tau$, guarda coordenadas $[i, j, \lambda_2]$ de p em uma lista L
- ③ Ordena L na ordem crescente de λ_2
- ④ Para cada ponto p , na ordem crescente de λ_2 , remove vizinhos de p em L .

Detecção de cantos

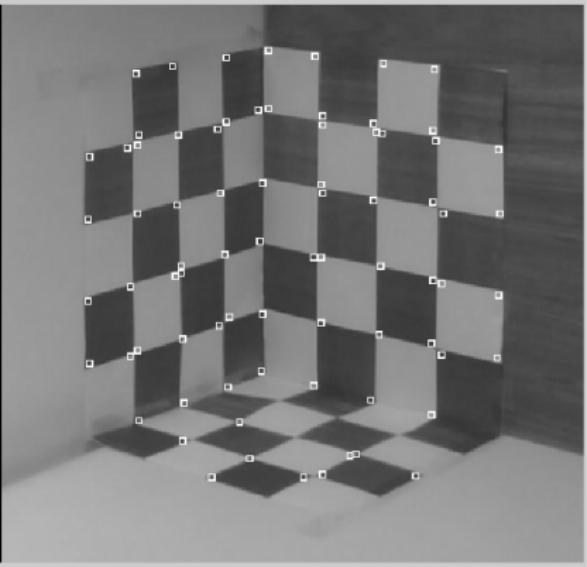
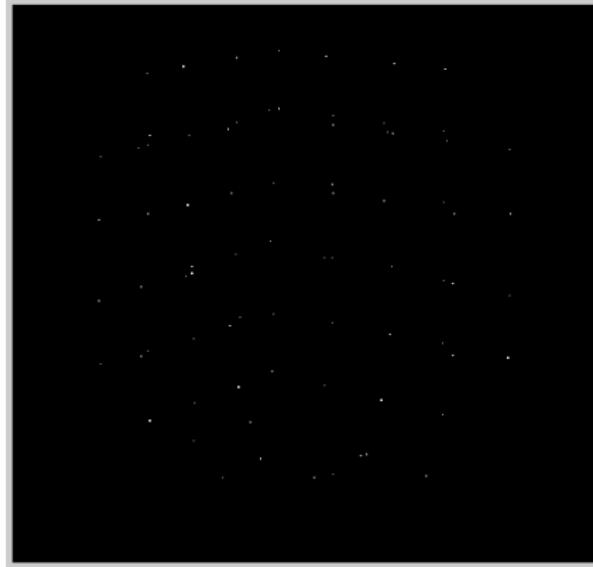
Exemplo de aplicação da detecção de cantos



```
» [S A]=corner('serro.bmp',2);
```

Detecção de cantos

Exemplo de aplicação da detecção de cantos



```
» [S A]=corner('canto.png',4);
```

Detecção de cantos

Um código para a detecção de cantos

```
function [S A]=corner(image,n)
A=imread(image);
A=rgb2gray(A);
[l,c]=size(A);
B=A;

DX=filtrolinear(A,[-1 -2 -1;  0 0 0;  1 2 1]);
DY=filtrolinear(A,[-1 0 1; -2 0 2; -1 0 1]);

L=[];
S=A*0;

for i=n+2:l-n
    for j=n+2:c-n
        C=[];
        C(:,:,1)=DX(i-n:i+n,j-n:j+n);
        C(:,:,2)=DY(i-n:i+n,j-n:j+n);
        C=reshape(C,(2*n+1)^2,2);
        e=min(eig(C'*C));
        if (e>300000)
            L=[L;[e,i,j]];
            S(i,j)=1;
        end
    end
end

P=sortrows(L,1);

for k=1:size(P)
    i=P(k,2);
    j=P(k,3);
    S(i-3:i+3,j-3:j+3)=zeros(7,7);
    S(i,j)=255;
end

W=ones(5,5)*255;
W(2:4,2:4)=0;
W=uint8(W);
for i=n+2:l-n
    for j=n+2:c-n
        if (S(i,j)>0)
            B=A(i-2:i+2,j-2:j+2);
            A(i-2:i+2,j-2:j+2)=B+W;
        end
    end
end

imshow([S, A]);
```

Detecção de curvas

Objetos na imagem são dados na forma:

- **Matricial:** Um conjunto de pixels descrevem um objeto, linha, polígono, etc;
- **Vetorial:** Uma descrição matemática descreve um objeto por sua equação e parâmetros;

Uma vez detectados pontos de aresta, na forma matricial, um passo importante é determinar descrição vetorial, com equações que os descrevem. Isso é feito na forma de:

- **Agrupamento:** Agrupa conjunto de pontos conforme a curva a que pertencem;
- **Ajuste de curvas:** Dado um conjunto de pontos, de uma dada curva, obter os parâmetros da curva;

Transformada de Hough

A transformada de **Hough** é um método de **agrupamento** de linhas.

Uma transformação do espaço da imagem para o espaço das linhas (ou espaço dos parâmetros) permite identificar as linhas.

- Cada linha $y = ax + b$ é identificada unicamente pelos parâmetros (a, b) . Então, uma linha é um ponto no espaço dos parâmetros.;
- Cada ponto $p = [x, y]$ na imagem intercepta as linhas $b = x(-a) + y$ com a e b variando para representar todas as possibilidades.

Para cada ponto $[x, y]$ na imagem, os respectivos (a, b) , tais que $y = ax + b$, recebem um voto, no espaço dos parâmetros.

Os pontos com maior votação no espaço dos parâmetros determinam agrupamentos de linhas da imagem.

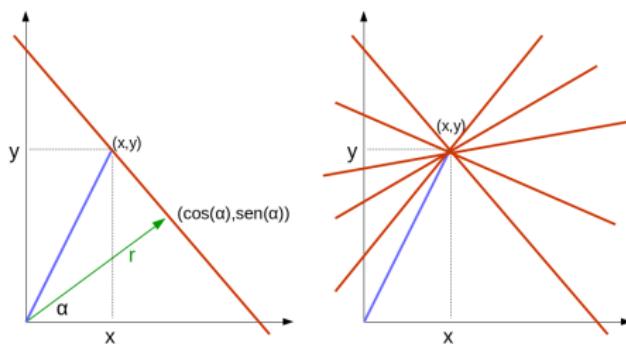
Transformada de Hough

Na forma explícita da reta $y = ax + b$, os valores de a variam de $-\infty$ até $+\infty$.

Ainda que usemos uma discretização do intervalo $(-\infty, +\infty)$, teríamos infinitos segmentos.

Na forma paramétrica, um ângulo α e um raio r definem a mesma reta como o conjunto dos pontos (x, y) tais que a projeção do vetor (x, y) sobre $(\cos(\alpha), \sin(\alpha))$ é igual a r .

Ou seja, $r = x * \cos(\alpha) + y * \sin(\alpha)$. Variando α , obtém-se os diversos r que definem as diversas retas que passam por (x, y) .



Transformada de Hough

Entrada: Uma imagem binária $E_{M \times N}$, onde $E(i, j) = 0$ ou $E(i, j) = 1$; Valor T com discretização do ângulo, $t \in [0, \pi]$; Valor R com discretização do raio, $r \in [0, \sqrt{M^2 + N^2}]$.

- 1 Discretiza ângulo T e raio R , obtendo dt e dr
- 2 Inicializa $A_{R \times T}$ com zero, para representar espaço dos parâmetros e receber votos
- 3 Para cada pixel $E(i, j)$, onde $E(i, j) = 1$, percorra $t = 1 \dots T$ e faça:
 - 1 $r = i * \cos(t * dt) + j * \sin(t * dt)$
 - 2 $r = r/dr$
 - 3 $A(r, t) = A(r, t) + 1$
- 4 Encontre os máximos locais em A

Transformada de Hough

Um código para a **Transformada de Hough**

```
function A=hough(E,R,T)

[M N]=size(E);

d=floor(sqrt(M^2+N^2))
dr=2*d/R
dt=pi/R

A=zeros(R,T);

for i=1:M
    for j=1:N
        for t=1:T
            if (E(i,j)>0)
                r=i*cos(t*dt)+j*sin(t*dt);
                r=round((r+d)/dr)+1;
                if (r>0 & r<R)
                    A(r,t)=A(r,t)+1;
                end
            end
        end
    end
end

C=showcurve(E,A,dr,dt,2);
```

Transformada de Hough

Exemplo de aplicação da transformada de Hough



```
» S=imread('linhas.png');  
» S=rgb2gray(S);  
» C=hough(S,200,200);
```

Transformada de Hough

A mesma estratégia pode ser adaptada para detecção outros tipos de curvas, como círculos, elipses e cônicas em geral.

No caso de círculos, para cada ponto (x, y) na imagem, percorrem-se os possíveis valores de centro (a, b) e calcula-se o raio $r = \sqrt{(x - a)^2 + (y - b)^2}$. Nesse caso, o espaço dos parâmetros é tridimensional.

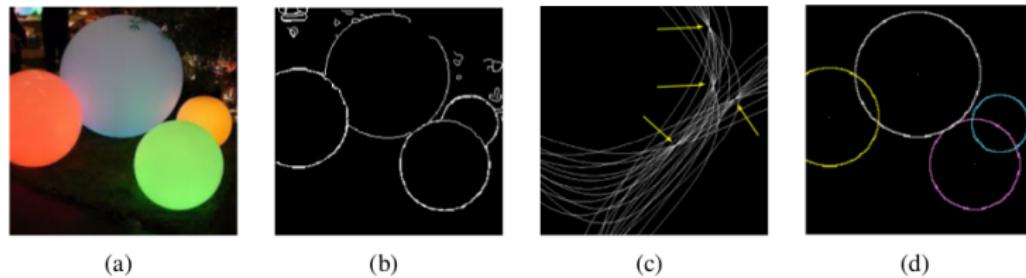
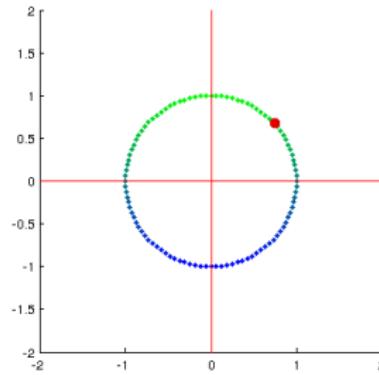


Figure 2- Passos na detecção de círculos usando a Transformada de Hough. (a) Imagem é suavizada usando filtro gaussiano. (b) As bordas são detectadas usando filtro de Canny. (c) A transformada faz a votação no espaço dos parâmetros (setas indicam máximos locais). (d) Usando os parâmetros indicados pelos máximos locais, os círculos são reconstruídos a partir do conhecimento do centro e raio.

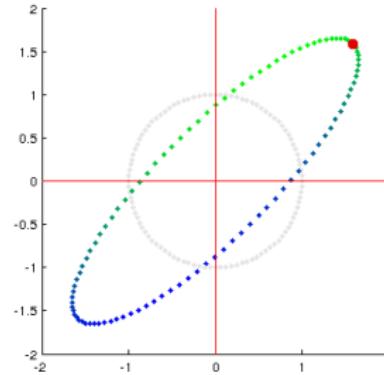
Ajuste de Curvas Conicas usando SVD

Autovalores e Autovetores:

Uma transformação linear T , em diversas situações, realiza apenas uma mudança de escala.



() P



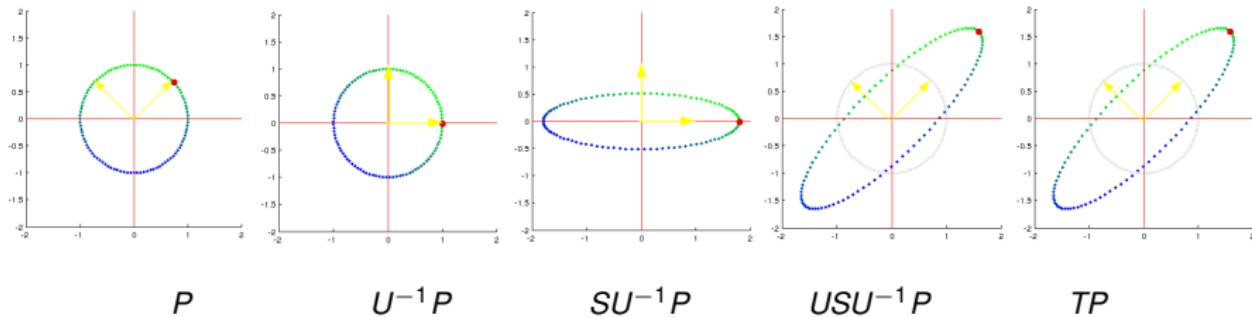
() TP

Porém, a mudança de escala, geralmente, não acontece ao longo dos eixos principais.

Ajuste de Curvas Conicas usando SVD

Autovalores e Autovetores:

Para que a mudança de escala seja percebida ao longo dos eixos principais, uma decomposição da transformação T é necessária.



Então, $T = USU^{-1}$ é uma decomposição de T , onde:

- U é a matriz cujas colunas são chamadas *autovetores* de T ;
- S é a matriz diagonal cujos elementos são *autovalores* de T .

Ajuste de Curvas Conicas usando SVD

Autovalores e Autovetores:

Definição: Um vetor $x \in R^n$ é dito *autovetor* para uma matriz quadrada $T_{n \times n}$, se existe $\lambda \in \mathbb{R}$ tal que

$$Tx = \lambda x \tag{1}$$

A decomposição $T = USU^{-1}$ obtém os autovetores como colunas de U e autovalores na diagonal de S .

- Restrito a matrizes quadradas $T_{n \times n}$;
- U não é, necessariamente, uma rotação (ortonormal).

Ajuste de Curvas Conicas usando SVD

Valores Singulares:

A Decomposição em Valores Singulares (SVD), busca decompor uma matriz $A_{m \times n}$ em rotações e escalas:

$$A = USV^t \quad (2)$$

- Matrizes $A_{m \times n}$, mesmo com $m \neq n$;
- $U_{m \times m}$ e $V_{n \times n}$ são matrizes de rotação (ortonormais).

$$\boxed{A} = \boxed{U} X \boxed{\begin{matrix} \diagdown \\ S \end{matrix}} X \boxed{V^t}$$

Ajuste de Curvas Conicas usando SVD

Valores Singulares:

Como computar U e V , ortonormais, tais que

$$A = USV^t \quad (3)$$

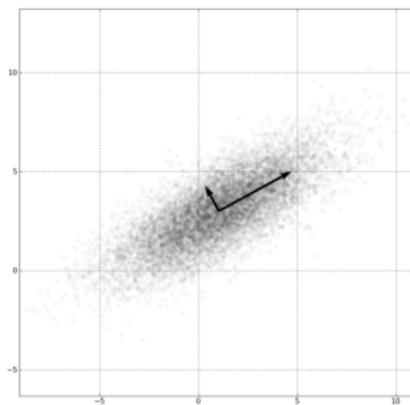
Note que:

- $A^t A = (VS^t U^t)(USV^t) = VS^t SV^t = VS^2 V^t$;
- $AA^t = (USV^t)(VS^t U^t) = USS^t U^t = US^2 U^t$.

Desta forma,

- V é a matriz de autovetores de $A^t A$
- U é a matriz de autovetores de AA^t
- S tem elementos como raiz de autovalores de $A^t A$ e AA^t

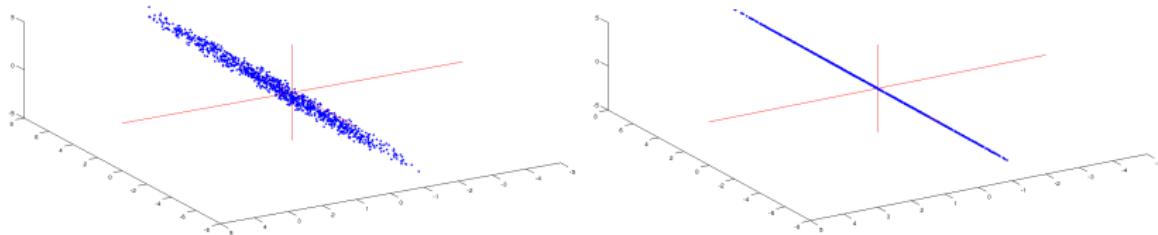
Decomposição em Valores Singulares (SVD)



Interpretação prática da Decomposição SVD ($A = USV^t$)

- U “explica” a distribuição de pontos nas colunas de A ;
- V “explica” a distribuição de pontos nas linhas de A ;
- S contém, na diagonal, a magnitude dos elementos da base.

Aplicações de SVD: Filtragem de ruído



O arquivo **nuvem.m** constroi uma nuvem de pontos A. Vamos encontrar as direções principais e remover a de menor variação.

```
» nuvem  
» plot3(A(1,:),A(2,:),A(3,:),'.' );  
» [u s v]=svd(A);  
» B=u(:,1:2)*s(1:2,1:2)*v(:,1:2)';  
» plot3(B(1,:),B(2,:),B(3,:),'.' );
```

Aplicações de SVD: Filtragem de ruído

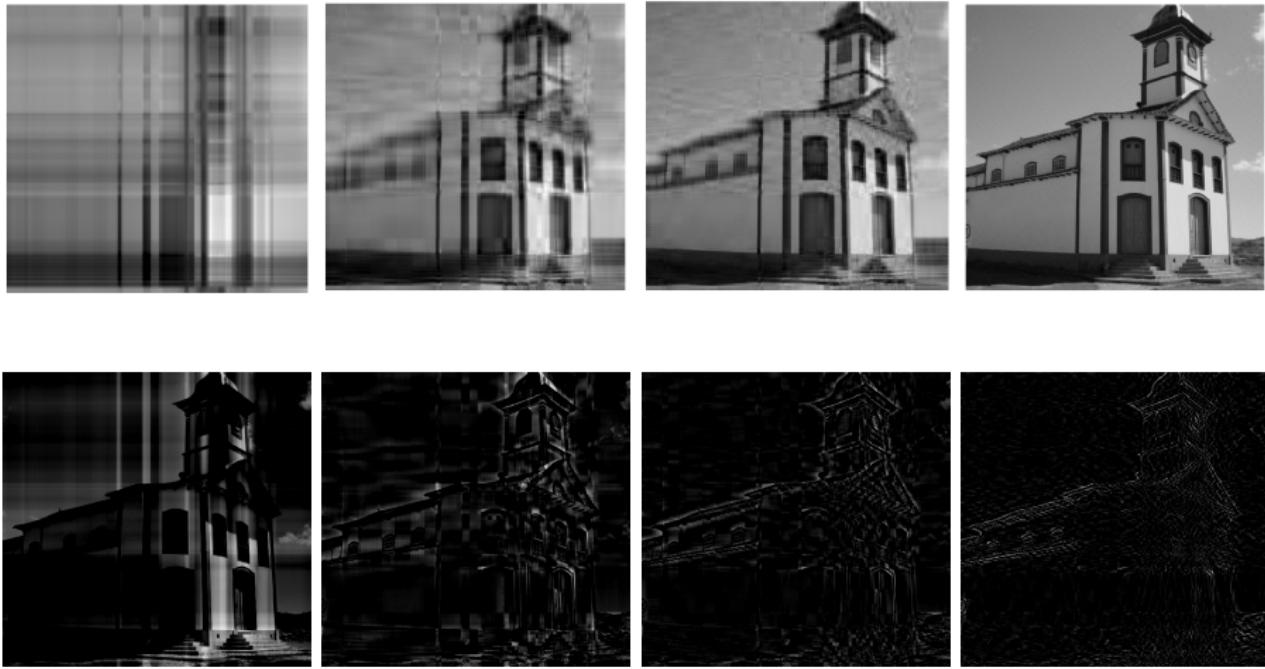
Na filtragem de ruído usando SVD os menores valores singulares são ignorados.

Isto implica que apenas as “componentes” associadas aos maiores valores singulares são utilizados.

Aplicações de SVD: Compressão de imagem



Aplicações de SVD: Compressão de imagem



$$[u \ s \ v] = \text{svd}(A);$$

$$B = u(:, k_1 : k_2) * s(k_1 : k_2, k_1 : k_2) * v(:, k_1 : K_2)'$$

Aplicações de SVD: Compressão de imagem

Na compressão de imagem usando SVD os menores valores singulares são ignorados.

Isto implica que apenas as “componentes” associadas aos maiores valores singulares são utilizados.

Aplicações de SVD: PCA (Principal Component Analisys)

Lembre que, ao decompor a matriz A em:

$$A = U \times S \times V'$$

A matriz U é uma matriz de rotação, de forma que fazendo

$$B = U' \times A$$

apenas rotacionamos elementos (colunas) de A para uma nova base que melhor “explica” a distribuição dos pontos em colunas de A .

Aplicações de SVD: PCA (Principal Component Analisys)

Na **Análise de Componentes Principais**, a base $U_{m \times m}$ é truncada para $U_{m \times k}$, com $k < m$, para formar uma base usando apenas as k componentes associadas aos k maiores valores singulares.

$$\begin{matrix} B_{kn} \\ \hline \end{matrix} = \begin{matrix} U'_{mk} \\ \hline \end{matrix} X \begin{matrix} A \\ \hline \end{matrix}$$

Esta projeção no **subespaço** associado às componentes principais leva a

- redução de dimensionalidade, que pode ser sem perdas;
- Remoção de ruído.

Geralmente, as novas “features” em B são usadas no reconhecimento de padrões.

Curvas cônicas

Usaremos decomposição **SVD** para fazer ajuste de curvas.

As cônicas são dadas na forma implícita por

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

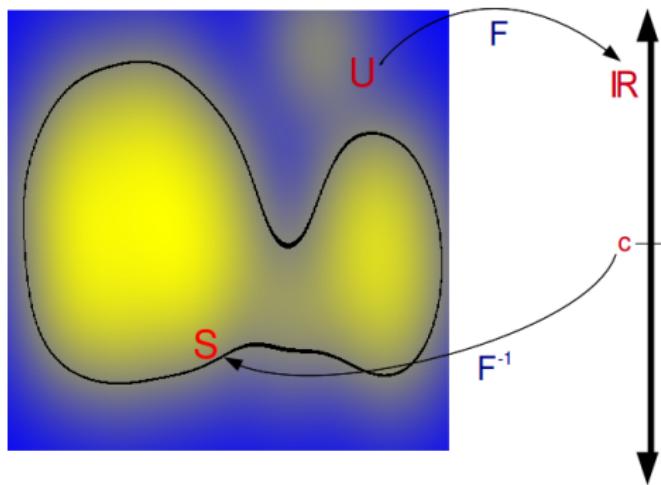
Conforme os valores de A, B, C, D, E, F a cônica pode definir:

- Ponto;
- Retas;
- Círculo;
- Elipse;
- Parábola;
- Hipérbole.

Curvas cônicas

Cônicas são **curvas implícitas** em \mathbb{R}^2 :

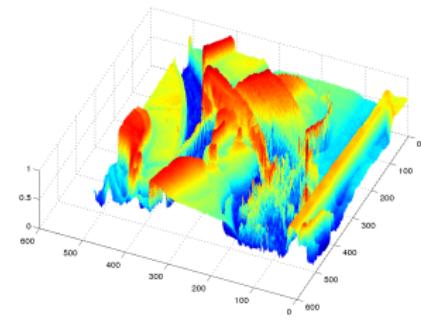
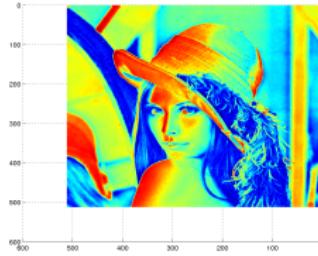
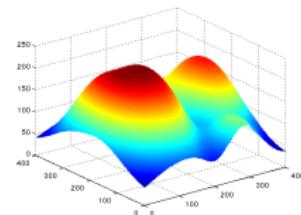
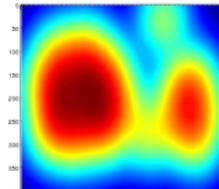
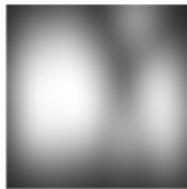
Um subconjunto $S \subset \mathbb{R}^2$ é uma curva implícita se existe uma função $F : U \rightarrow \mathbb{R}$, $S \subset U$ e um número $c \in \mathbb{R}$ tal que $S = F^{-1}(c)$.



S é regular se F é diferenciável e $\nabla F(p) \neq 0$ para todo $p \in S$.

Curvas cônicas

Diferentes curvas de nível são definidas para um mesmo campo escalar dependendo do nível de corte, ou seja, do número $c \in \mathbb{R}$ tal que $S = F^{-1}(c)$.



Curvas cônicas

Polinômio em \mathbb{R}^2 :

Um polinômio de grau d definido em \mathbb{R}^2 é uma função $P_d : \mathbb{R}^2 \rightarrow \mathbb{R}$ dada pela expressão:

$$P_d(x, y) = \sum_{0 \leq i+j \leq d} a_{i,j} x^i y^j$$

Curva Algébrica:

Uma curva algébrica de grau d é a curva implícita

$$S = P_d^{-1}(0)$$

Curvas cônicas

Notação para P_d :

Adotaremos uma notação vetorial para expressar um polinômio P_d :

$$P_d(x, y) = P^t Q,$$

onde os elementos do vetor P são os monômios de P_d e os elementos do vetor Q são os coeficientes do polinômio.

Exemplo:

$$P_2(x, y) = 2x^2 + xy - 2x + 2y + 4$$

$$P^t = [x^2, y^2, xy, x, y, 1] \quad Q^t = [2, 0, 1, -2, 2, 4]$$

Curvas cônicas

Uma classe particular de curvas algébricas são cônicas, $f^{-1}(0)$, onde:

$$f = P_2(x, y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$$

Escrevendo

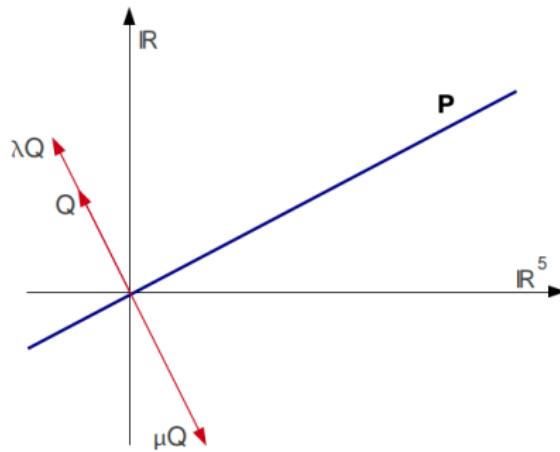
$$Q = [A, B, C, D, E, F]^t \text{ e}$$

$$P = [x^2, y^2, xy, x, y, 1]^t$$

temos f como o produto escalar $f(x, y) = P^t \cdot Q$.

Curvas cônicas

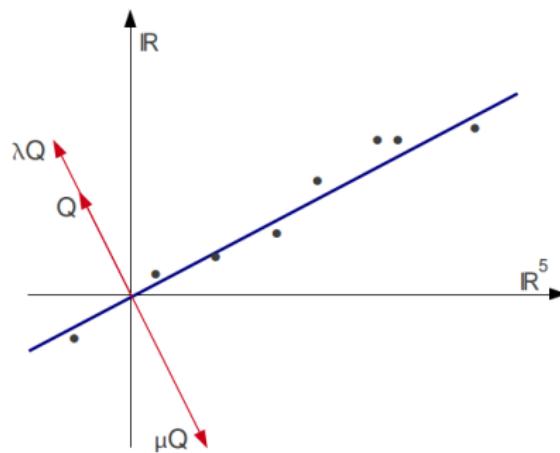
Usando essa notação, uma cônica fica bem definida pelo seu vetor de coeficientes Q ou, mais precisamente, por uma classe $[Q]$ no espaço projetivo $P(\mathbb{R}^6)$.



$$P^t \cdot Q = 0.$$

Curvas cônicas

Dado um conjunto de pontos $\{p_i = (x_i, y_i), i = 1..n\}$, temos que p_i está mais próximo de uma cônica de coeficientes Q à medida que $f(x_i, y_i) = P_i^t \cdot Q$ se aproxima de 0.



Então, a cônica que melhor se ajusta ao conjunto de pontos tem coeficientes Q que minimizam $\sum(P_i^t \cdot Q)^2$.

Curvas cônicas

Temos, por associatividade, que:

$$\sum(P_i^t.Q)^2 = \sum(P_i^t.Q.P_i^t.Q) = Q^t \left(\sum P_i P_i^t \right) Q$$

Indicando por A a matriz $A = \sum P_i P_i^t$, temos $\sum(P_i^t.Q)^2 = Q^t.A.Q$

Então, a cônica que melhor se ajusta ao conjunto de pontos tem coeficientes Q que minimizam

$$Q^t.A.Q$$

Como a matriz A é um operador linear simétrico, existe uma base ortonormal, dada pelos autovetores de A .

Então, a cônica de melhor ajute Q é dada pelo autovetor associado ao menor dos autovalores de A .

Curvas cônicas

Exemplo de implementação do ajuste a elipse

```
function [a b r1 r2]=elipse(S)
[l c]=size(S);
P=[];
for x=1:l
    for y=1:c
        if (S(x,y)>0)
            P=[P; [x^2, y^2, x*y, x, y, 1]];
        end
    end
end

[u s v] = svd(P'*P);
A=u(1,6); B=u(2,6); C=u(3,6); D=u(4,6); E=u(5,6); F=u(6,6);

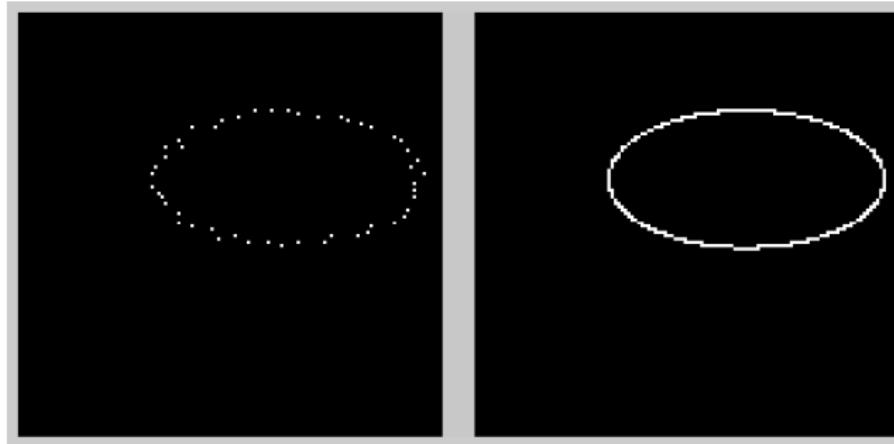
a = - D / (2*A)
b = - E / (2*B)
r1 = sqrt((a^2*A + b^2*B - F)/A)
r2 = sqrt((a^2*A + b^2*B - F)/B)

H=S*0;
for t=0:0.01:2*pi
    x=round(a+sin(t)*r1);
    y=round(b+cos(t)*r2);
    H(x,y)=255;
end

imshow([S ones(l,10)*200 H]);
```

Curvas cônicas

Exemplo de uso do programa **elipse.m**



```
» S=imread('elipse.png');  
» [a b r1 r2]=elipse(S);
```

Curvas cônicas

Dado um conjunto de pontos (x_i, y_i) descrevendo uma cônica qualquer, podemos obter A, B, C, D, E, F tais que

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

descreva a equação da cônica de melhor ajuste aos pontos, independente de ser **círculo, elipse, reta, parábola ou hipérbole**.

```
function [A B C D E F]=conica(x,y)
n = size(x,2);
P=[];
for i=1:n
    P=[P; [x(i)^2, y(i)^2, x(i)*y(i), x(i), y(i), 1]];
end
[U S V] = svd(P'*P);
A=U(1,6); B=U(2,6); C=U(3,6); D=U(4,6); E=U(5,6); F=U(6,6);
```

Curvas cônicas

Como a cônica é dada implicitamente, nos parâmetros A, B, C, D, E, F , precisamos de uma forma de vizualizar a curva, implicitamente, sem saber do que se trata, **círculo, elipse, reta, parábola ou hipérbole**.

```
function C=showconica(image)

S=imread(image);
[m n k]=size(S)
if (k>1); S=rgb2gray(S); end;

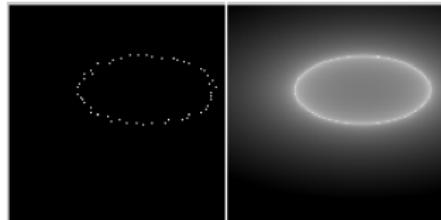
X=[]; Y=[];
for i=1:m
    for j=1:n
        if (S(i,j)>0)
            X=[X; i];
            Y=[Y; j];
        end
    end
end

[a b c d e f]=conica(X',Y');

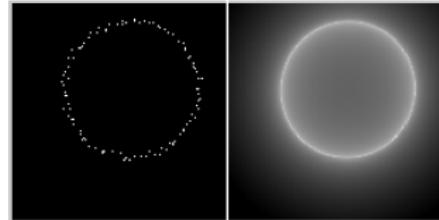
C=[];
for x=1:m; for y=1:n
    C(x,y) = ( a*x^2 + b*y^2 + c*x*y + d*x + e*y + f )^(0.2);
end; end

imshow([S,(1-C)*255]);
```

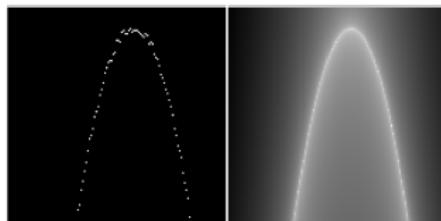
Curvas cônicas



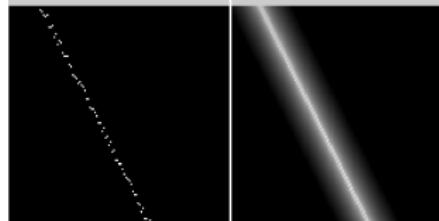
```
» C=showconica('elipse.png');
```



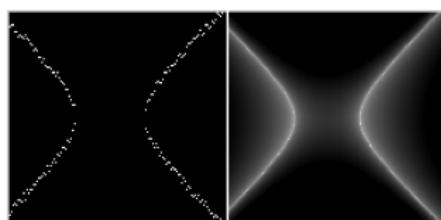
```
» C=showconica('circulo.png');
```



```
» C=showconica('parabola.png');
```



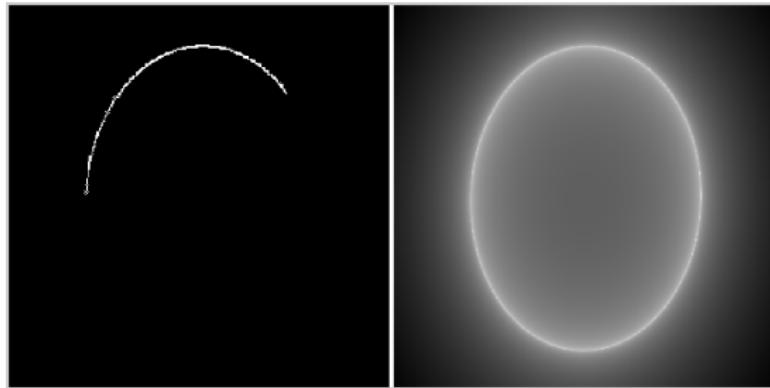
```
» C=showconica('reta.png');
```



```
» C=showconica('hiperbole.png');
```

Curvas cônicas

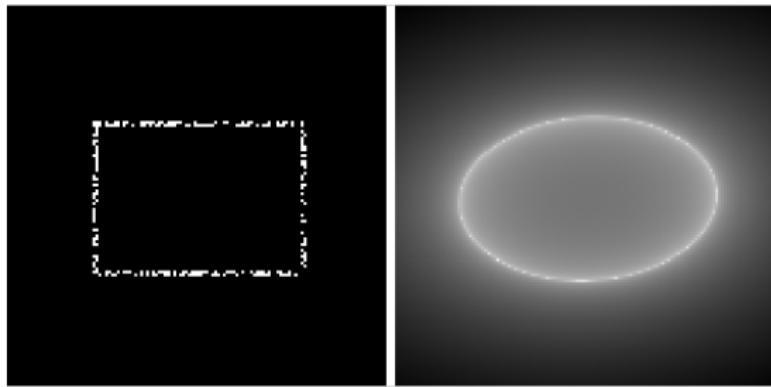
Teoricamente, qualquer segmento da curva é suficiente para fazer sua reconstrução.



```
» C=showconica('parcial.png');
```

Curvas cônicas

Qualquer conjunto de pontos pode ser ajustado por uma cônica, ainda que isso não faça sentido na prática.



```
» C=showconica('square.png');
```

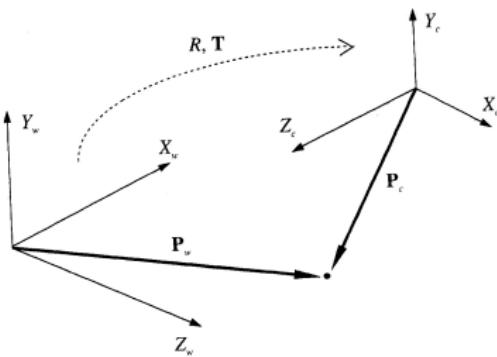
Parâmetros da câmera

A utilidade da *Visão Computacional* está em associar **pixels** da imagem com **coordenadas 3D** de pontos da cena.

Essa associação depende do conhecimento de um conjunto de parâmetros **intrínsecos** e **extrínsecos**.

- Os parâmetros **Extrínsecos** definem a localização e orientação da câmera com respeito a um sistema de coordenadas conhecido do mundo;
- Os parâmetros **Intrínsecos** a definem a transformação de um ponto no sistema de coordenadas da câmera para coordenadas de pixel numa imagem.

Parâmetros da câmera: Extrínsecos



- Matriz de rotação $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$;
- Vetor de translação $T = [T_x, T_y, T_z]$.

R e T , chamados **parâmetros extrínsecos**, são usados para estabelecer a relação

$$P_c = R.P_w + T$$

Parâmetros da câmera: Intrínsecos

Os parâmetros *intrínsecos* são:

- A distância focal f ;
- O tamanho do pixel, em milímetros, s_x e s_y
- O centro da imagem (o_x , o_y)

As coordenadas, em pixels $p_{im} = [x_{im}, y_{im}]$, são obtidas das coordenadas $P_c = [X_c, Y_c, Z_c]$ no sistema de referência da câmera fazendo

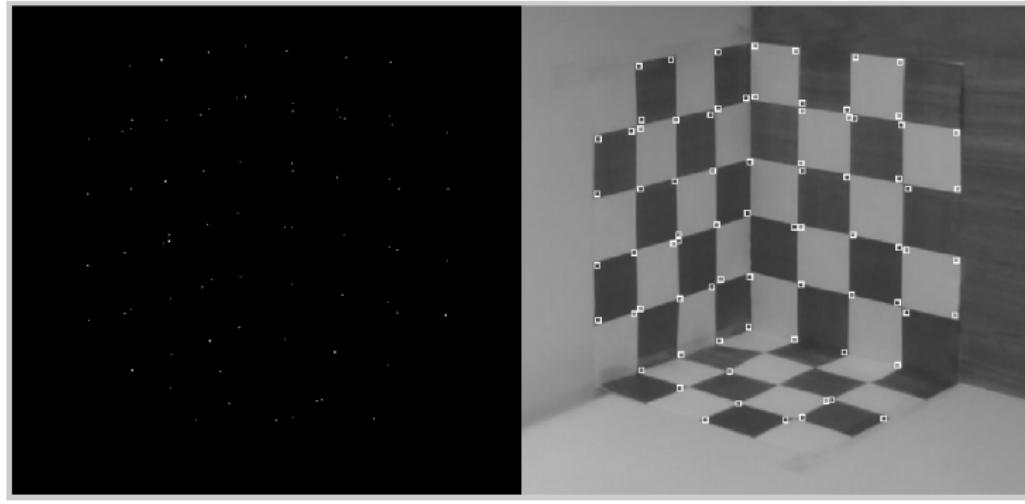
$$x_{im} = o_x - \frac{f}{s_x} \frac{X_c}{Z_c} \quad \text{e} \quad y_{im} = o_y - \frac{f}{s_y} \frac{Y_c}{Z_c}$$

Na forma matricial, os parâmetros intrínsecos definem a matriz M_{int} , dada por

$$M_{int} = \begin{bmatrix} \frac{-f}{s_x} & 0 & o_x \\ 0 & \frac{-f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix}.$$

Calibração de câmeras

O algoritmo de **Calibração Direta** usa um conjunto n de pontos $\{(X_i^w, Y_i^w, Z_i^w), i = 1..n\}$ conhecidos em coordenadas do mundo, e suas respectivas coordenadas na imagem $\{(x_i, y_i), i = 1..n\}$ para estimar os parâmetros extrínsecos e intrínsecos.



Calibração de câmeras

Vimos que a associação de cada (X^w, Y^w, Z^w) com sua imagem (x, y) passa pelas coordenadas da câmera (X^c, Y^c, Z^c) .

$$(X^w, Y^w, Z^w) \Rightarrow (X^c, Y^c, Z^c) \Rightarrow (x, y)$$

Os parâmetros extrínsecos (\mathbf{R}, \mathbf{T}) associam cada (X^w, Y^w, Z^w) com respectivos (X^c, Y^c, Z^c) .

$$\begin{pmatrix} X^c \\ Y^c \\ Z^c \end{pmatrix} = R \begin{pmatrix} X^w \\ Y^w \\ Z^w \end{pmatrix} + T = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}.$$

Em componentes, temos

$$\begin{cases} X^c = r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x \\ Y^c = r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y \\ Z^c = r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z \end{cases}$$

Calibração de câmeras

Os parâmetros intrínsecos (f, o_x, o_y, s_x, s_y) associam os pontos em coordenadas da câmera (X^c, Y^c, Z^c) com pontos da imagem (x, y)

$$\begin{cases} x = -\frac{f}{s_x} \frac{X_c}{Z_c} + o_x \\ y = -\frac{f}{s_y} \frac{Y_c}{Z_c} + o_y \end{cases}$$

Escrevendo $f_x = \frac{f}{s_x}$ e $f_y = \frac{f}{s_y}$ e assumindo $(o_x, o_y) = (0, 0)$, temos

$$\begin{cases} x = -f_x \frac{X_c}{Z_c} \\ y = -f_y \frac{Y_c}{Z_c} \end{cases}$$

Calibração de câmeras

De $\begin{cases} X^c = r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x \\ Y^c = r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y \\ Z^c = r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z \end{cases}$ e $\begin{cases} x = -f_x \frac{X_c}{Z_c} \\ y = -f_y \frac{Y_c}{Z_c} \end{cases}$

escrevemos a coordenada (x, y) , da imagem, diretamente de (X^w, Y^w, Z^w) , sem usar (X^c, Y^c, Z^c)

$$\begin{cases} x = -f_x \frac{r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x}{r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z} \\ y = -f_y \frac{r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y}{r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z} \end{cases}$$

Calibração de câmeras

De

$$\begin{cases} x = -f_x \frac{r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x}{r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z} \\ y = -f_y \frac{r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y}{r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z} \end{cases}$$

multiplicamos a primeira equação por y e a segunda por x , obtendo

$$\begin{cases} xy = -y.f_x \cdot \frac{r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x}{r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z} \\ xy = -x.f_y \cdot \frac{r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y}{r_{31}X^w + r_{32}Y^w + r_{33}Z^w + T_z} \end{cases}$$

E, portanto, temos a equação

$$x.f_y(r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y) = y.f_x(r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x)$$

ou

$$x.(r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y) = y \cdot \frac{f_x}{f_y} (r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x)$$

Calibração de câmeras

De

$$x.(r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y) = y \cdot \frac{f_x}{f_y} (r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x)$$

e fazendo $\alpha = \frac{f_x}{f_y}$, temos

$$x.(r_{21}X^w + r_{22}Y^w + r_{23}Z^w + T_y) = y.\alpha(r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x)$$

Finalmente, tomamos $\left\{ \begin{array}{ll} \mathbf{a} = r_{21} & \mathbf{e} = \alpha r_{11} \\ \mathbf{b} = r_{22} & \mathbf{f} = \alpha r_{12} \\ \mathbf{c} = r_{23} & \mathbf{g} = \alpha r_{13} \\ \mathbf{d} = T_y & \mathbf{h} = \alpha T_x \end{array} \right\}$, e temos

$$xX^w\mathbf{a} + xY^w\mathbf{b} + xZ^w\mathbf{c} + x\mathbf{d} = yX^w\mathbf{e} + yY^w\mathbf{f} + yZ^w\mathbf{g} + y\mathbf{h}$$

$$\Rightarrow xX^w\mathbf{a} + xY^w\mathbf{b} + xZ^w\mathbf{c} + x\mathbf{d} - yX^w\mathbf{e} - yY^w\mathbf{f} - yZ^w\mathbf{g} - y\mathbf{h} = 0$$

Calibração de câmeras

A equação

$$xX^W\mathbf{a} + xY^W\mathbf{b} + xZ^W\mathbf{c} + x\mathbf{d} - yX^W\mathbf{e} - yY^W\mathbf{f} - yZ^W\mathbf{g} - y\mathbf{h} = 0$$

deve ser satisfeita para os n pares de pontos $\{(X_i^W, Y_i^W, Z_i^W), i = 1..n\}$ e $\{(x_i, y_i), i = 1..n\}$ previamente estabelecidos. Então

$$\left\{ \begin{array}{l} x_1 X_1^W \mathbf{a} + x_1 Y_1^W \mathbf{b} + x_1 Z_1^W \mathbf{c} + x_1 \mathbf{d} - y_1 X_1^W \mathbf{e} - y_1 Y_1^W \mathbf{f} - y_1 Z_1^W \mathbf{g} - y_1 \mathbf{h} = 0 \\ x_2 X_2^W \mathbf{a} + x_2 Y_2^W \mathbf{b} + x_2 Z_2^W \mathbf{c} + x_2 \mathbf{d} - y_2 X_2^W \mathbf{e} - y_2 Y_2^W \mathbf{f} - y_2 Z_2^W \mathbf{g} - y_2 \mathbf{h} = 0 \\ \vdots \quad \vdots \\ x_n X_n^W \mathbf{a} + x_n Y_n^W \mathbf{b} + x_n Z_n^W \mathbf{c} + x_n \mathbf{d} - y_n X_n^W \mathbf{e} - y_n Y_n^W \mathbf{f} - y_n Z_n^W \mathbf{g} - y_n \mathbf{h} = 0 \end{array} \right.$$

$$\left[\begin{array}{cccccccc} x_1 X_1^W & x_1 Y_1^W & x_1 Z_1^W & x_1 & -y_1 X_1^W & -y_1 Y_1^W & -y_1 Z_1^W & -y_1 \\ x_2 X_2^W & x_2 Y_2^W & x_2 Z_2^W & x_2 & -y_2 X_2^W & -y_2 Y_2^W & -y_2 Z_2^W & -y_2 \\ \vdots & \vdots \\ x_n X_n^W & x_n Y_n^W & x_n Z_n^W & x_n & -y_n X_n^W & -y_n Y_n^W & -y_n Z_n^W & -y_n \end{array} \right] \left[\begin{array}{c} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \\ \mathbf{e} \\ \mathbf{f} \\ \mathbf{g} \\ \mathbf{h} \end{array} \right] = \left[\begin{array}{c} 0 \\ 0 \\ \vdots \\ 0 \end{array} \right]$$

Calibração de câmeras

$$\begin{bmatrix} x_1 X_1^W & x_1 Y_1^W & x_1 Z_1^W & x_1 & -y_1 X_1^W & -y_1 Y_1^W & -y_1 Z_1^W & -y_1 \\ x_2 X_2^W & x_2 Y_2^W & x_2 Z_2^W & x_2 & -y_2 X_2^W & -y_2 Y_2^W & -y_2 Z_2^W & -y_2 \\ \vdots & \vdots \\ x_n X_n^W & x_n Y_n^W & x_n Z_n^W & x_n & -y_n X_n^W & -y_n Y_n^W & -y_n Z_n^W & -y_n \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \\ \mathbf{e} \\ \mathbf{f} \\ \mathbf{g} \\ \mathbf{h} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

O sistema será escrito como $\mathbf{Av} = \mathbf{0}$, onde o vetor de incógnitas

$$\mathbf{v} = (\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h})$$

será usado para se obter os parâmetros extrínsecos e intrínsecos.

A matriz \mathbf{A} tem posto 7. Então, se $n \geq 7$, o sistema homogêneo $\mathbf{Av} = \mathbf{0}$ tem solução não trivial.

Fazendo a decomposição em valores singulares, $\mathbf{A} = \mathbf{UDV}^t$, teremos que \mathbf{v} será o autovetor de \mathbf{V} associado ao menor dos autovalores de \mathbf{A} .

Calibração de câmeras

$$\begin{bmatrix} x_1 X_1^w & x_1 Y_1^w & x_1 Z_1^w & x_1 & -y_1 X_1^w & -y_1 Y_1^w & -y_1 Z_1^w & -y_1 \\ x_2 X_2^w & x_2 Y_2^w & x_2 Z_2^w & x_2 & -y_2 X_2^w & -y_2 Y_2^w & -y_2 Z_2^w & -y_2 \\ \vdots & \vdots \\ x_n X_n^w & x_n Y_n^w & x_n Z_n^w & x_n & -y_n X_n^w & -y_n Y_n^w & -y_n Z_n^w & -y_n \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \\ \mathbf{e} \\ \mathbf{f} \\ \mathbf{g} \\ \mathbf{h} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Para construir o sistema $\mathbf{Av} = \mathbf{0}$, acima, a partir de n pares de pontos $\{(X_i^w, Y_i^w, Z_i^w), i = 1..n\}$ e $\{(x_i, y_i), i = 1..n\}$, é importante observar:

- O número de pontos deve ser $n \geq 7$;
- Devido a ruído na aquisição dos pontos, \mathbf{A} pode não ter autovalor nulo;
- O menor autovalor deve ser considerado na solução;
- O vetor \mathbf{v} solução existe a menos de um fator de escala γ .

Então obtemos uma solução $\bar{\mathbf{v}} = \gamma \mathbf{v} = \gamma(r_{21}, r_{22}, r_{23}, T_y, \alpha r_{11}, \alpha r_{12}, \alpha r_{13}, \alpha T_x)$

Calibração de câmeras

De $\bar{\mathbf{v}} = (\gamma r_{21}, \gamma r_{22}, \gamma r_{23}, \gamma T_y, \gamma \alpha r_{11}, \gamma \alpha r_{12}, \gamma \alpha r_{13}, \gamma \alpha T_x)$, temos

- Sabendo que $r_{21}^2 + r_{22}^2 + r_{23}^2 = 1$, temos que

$$\sqrt{\bar{v}_1^2 + \bar{v}_2^2 + \bar{v}_3^2} = \sqrt{\gamma^2(r_{21}^2 + r_{22}^2 + r_{23}^2)} = |\gamma|$$

- Da mesma forma, $r_{11}^2 + r_{12}^2 + r_{13}^2 = 1$, então

$$\sqrt{\bar{v}_5^2 + \bar{v}_6^2 + \bar{v}_7^2} = \sqrt{\gamma^2 \alpha^2(r_{11}^2 + r_{12}^2 + r_{13}^2)} = |\gamma|\alpha$$

Então, temos

$$\begin{cases} (r_{21}, r_{22}, r_{23}) = \frac{1}{\gamma}(\bar{v}_1, \bar{v}_2, \bar{v}_3) \\ (r_{11}, r_{12}, r_{13}) = \frac{1}{\gamma\alpha}(\bar{v}_5, \bar{v}_6, \bar{v}_7) \end{cases} \quad \text{e} \quad \begin{cases} T_x = \frac{1}{\gamma}\bar{v}_4 \\ T_y = \frac{1}{\alpha\gamma}\bar{v}_8 \end{cases}$$

De $(r_{31}, r_{32}, r_{33}) = (r_{11}, r_{12}, r_{13}) \wedge (r_{21}, r_{22}, r_{23})$, temos $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$

Calibração de câmeras

Devido a possível ruído, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$ obtido pode não ser ortonormal.

Para garantir a ortonormalidade, fazemos a decomposição em valores singulares $R = UDV^t$, e substituindo D pela identidade I , reconstruímos $R = UIV^t$.

- Como conhecemos $|\gamma|$, não sabemos o sinal de γ . Pra isso, observemos que x na imagem e X^c deve ter sinal contrário, ou seja $x.X^c < 0$

Então, se $x.X^c = x.(r_{11}X^w + r_{12}Y^w + r_{13}Z^w + T_x) > 0$, deve-se trocar os sinais de (T_x, T_y) e das linhas R_1 e R_2 de R .

Falta conhecer T_z e f_x para que a calibração esteja concluída!

Calibração de câmeras

Para obter T_z e f_x , observamos que, de $x_i = -f_x \frac{X_i^c}{Z_i^c} \Rightarrow x_i Z_i^c = -f_x X_i^c$, obtemos para todo $i = 1 \dots n$, que

$$x_i(r_{31}X_i^w + r_{32}Y_i^w + r_{33}Z_i^w + \mathbf{T}_z) = -\mathbf{f}_x(r_{11}X_i^w + r_{12}Y_i^w + r_{13}Z_i^w + T_x)$$

$$x_i\mathbf{T}_z + \mathbf{f}_x(r_{11}X_i^w + r_{12}Y_i^w + r_{13}Z_i^w + T_x) = -x_ir_{31}X_i^w - x_ir_{32}Y_i^w - x_ir_{33}Z_i^w$$

Chamando $\begin{cases} m_i = r_{11}X_i^w + r_{12}Y_i^w + r_{13}Z_i^w + T_x \\ p_i = -x_ir_{31}X_i^w - x_ir_{32}Y_i^w - x_ir_{33}Z_i^w \end{cases}$

temos o sistema $\begin{cases} x_1\mathbf{T}_z + \mathbf{f}_x m_1 = p_1 \\ x_2\mathbf{T}_z + \mathbf{f}_x m_2 = p_2 \\ \vdots \quad \vdots \quad \vdots \\ x_n\mathbf{T}_z + \mathbf{f}_x m_n = p_n \end{cases} \Rightarrow \begin{bmatrix} x_1 & m_1 \\ x_2 & m_2 \\ \vdots & \vdots \\ x_n & m_n \end{bmatrix} \begin{bmatrix} \mathbf{T}_z \\ \mathbf{f}_x \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}$

A solução desse sistema, por mínimos quadrados, leva aos valores de T_z e f_x .

Calibração de câmeras

Finalmente, de $\alpha = \frac{f_x}{f_y}$, temos $f_y = \frac{f_x}{\alpha}$.

Com o centro da imagem (o_x, o_y) conhecidos, temos todas as informações para recuperar M_{ext} e M_{int}

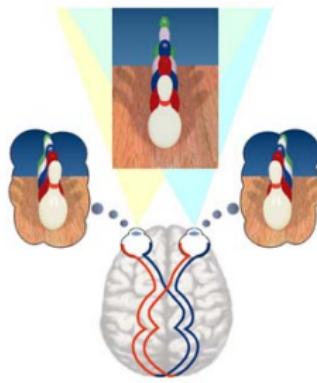
$$M_{ext} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & -R_1^T T \\ r_{21} & r_{22} & r_{23} & -R_2^T T \\ r_{31} & r_{32} & r_{33} & -R_3^T T \end{bmatrix} \quad \text{e} \quad M_{int} = \begin{bmatrix} \frac{-f}{s_x} & 0 & o_x \\ 0 & \frac{-f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix}.$$

Visão estéreo

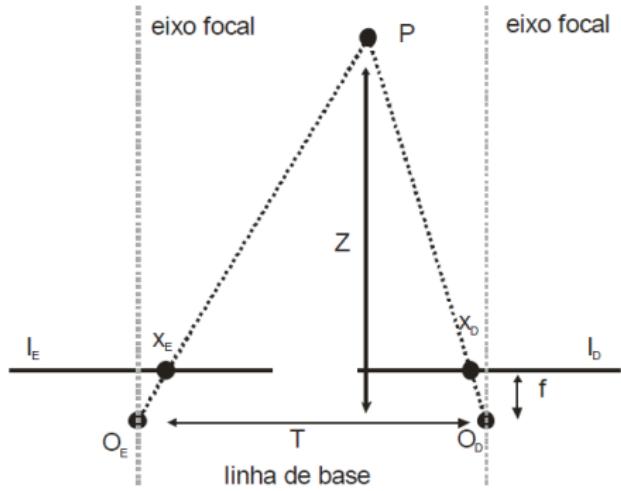
Usando duas ou mais câmeras de perspectiva, pode-se obter imagens da mesma cena de diferentes pontos de vista.

A *visão estéreo* é o processo de inferir informação tridimensional e profundidade em uma cena a partir de duas ou mais imagens.

A experiência humana de percepção de distâncias e profundidade é o exemplo mais imediato de *visão estéreo*. Cada olho captura uma imagem distinta e, a sobreposição de pontos correspondentes nessas imagens é processada no cérebro.



Visão estéreo



O deslocamento necessário para sobrepor pontos correspondentes será inversamente proporcional à profundidade do ponto.

$$\frac{T}{Z} = \frac{T - (x_E - x_D)}{Z - f} \Rightarrow \frac{T}{Z} = \frac{T - d}{Z - f} \Rightarrow Z = f \cdot \frac{T}{d}$$

rodovia se movendo rapidamente em relação ao carro, parece acompanhar a

Visão estéreo

O deslocamento **d** entre pontos correspondentes é chamado **disparidade**.

A relação das disparidades para todos os pontos da imagem é chamado **mapa de disparidade**.

Enquanto que, para o cérebro humano, parece relativamente simples processar as duas imagens e inferir o mapa de disparidade, do ponto de vista computacional é um trabalho complexo.

O problema de reconstrução 3D tenta obter um modelo tridimensional da cena a partir do **mapa de disparidades** e conhecimento da geometria do sistema.

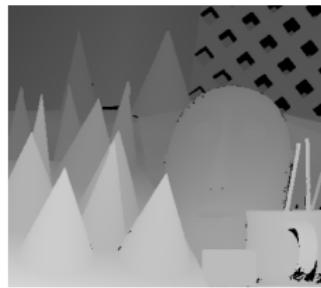
Visão estéreo

Do ponto de vista computacional, a visão estéreo deve resolver dois problemas:

- **Correspondência:** A que ponto, visto pela câmera da esquerda, corresponde cada ponto visto pela câmera da direita?



- **Reconstrução:** Dado um conjunto de pontos correspondentes, como obter a estrutura tridimensional da cena?



Visão estéreo

O problema de **Correspondência**:

Ao buscar correspondência, deve-se estabelecer que tipo de elementos buscar.

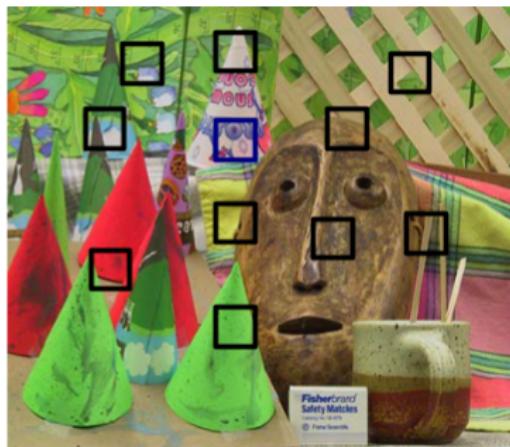


Pode-se buscar por pixels correspondentes, mas haveria muita redundância.

Pode-se buscar por elementos conhecidos nas imagens, mas isso requer conhecimento prévio.

Visão estéreo

A estratégia mais conveniente para correspondência é definir janelas de busca de tamanho $N \times N$ pixels.



Essa estratégia, chamada correlação, usa uma janela de tamanho fixo na imagem da esquerda e busca pela janela da direita que maximiza algum critério de similaridade.

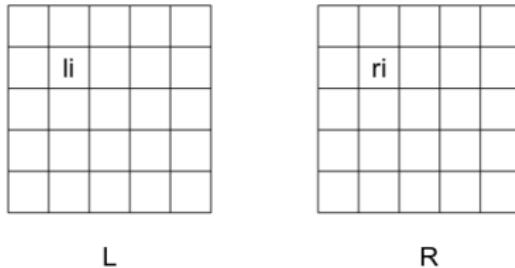
Visão estéreo

Idealmente, a busca não deve percorrer toda a imagem da direita, mas se restringir a uma linha nessa imagem.



Essa restrição é possível usando a chamada **restrição epipolar**, que veremos em seguida.

Visão estéreo



Dadas uma janela esquerda L e uma janela direita R para serem comparadas, alguma métrica deve ser adotada:

- $d(L, R) = \sum |l_i - r_i|$
- $d(L, R) = \max |l_i - r_i|$
- $d(L, R) = \sum (l_i - r_i)^2$
- $d(L, R) = - \sum l_i \cdot r_i$

Entre os possíveis correspondentes da direita, será escolhido aquele que minimiza o valor de $d(L, R)$.

Visão estéreo

Dadas imagens **E** e **D**, o algoritmo básico para computar disparidades tem os passos:

- Para cada pixel **E(i,j)**
 - ▶ Define uma janela **L**
 - ▶ Para cada pixel **D(i,k)**
 - ★ Define uma janela **R**
 - ★ Calcula $d(L,R)$
 - ▶ Obtem k que minimize $d(L,R)$
 - ▶ Faça $Disp(i,j) = |j - k|$
- Retorna **Disp**

Visão estéreo

Um código Matlab simples para cálculo da disparidade.

```
function Disp=disparidade(left,right,w)

E=imread(left);
D=imread(right);

[l c n]=size(E);

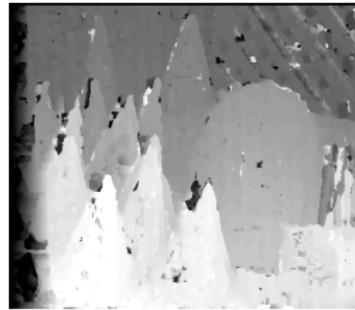
for i=w+1:l-w-1
    for j=w+1:c-w-1
        L=double(E(i-w:i+w,j-w:j+w,:));
        d=inf(1,c);
        for k=max(w+1,j-50):j
            R=double(D(i-w:i+w,k-w:k+w,:));
            d(k)=sum(sum(abs(L-R)));
        end
        [s t]=min(d);
        Disp(i,j)=abs(j-t);
    end
end

imshow(Disp/max(max(Disp))));
```

» Disp=disparidade('left.png','right.png',3);

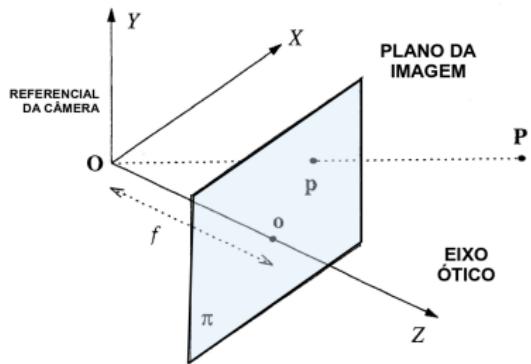
Visão estéreo

```
» Disp=disparidade('left.png','right.png',3);
```



GEOMETRIA EPIPOLAR

Geometria epipolar

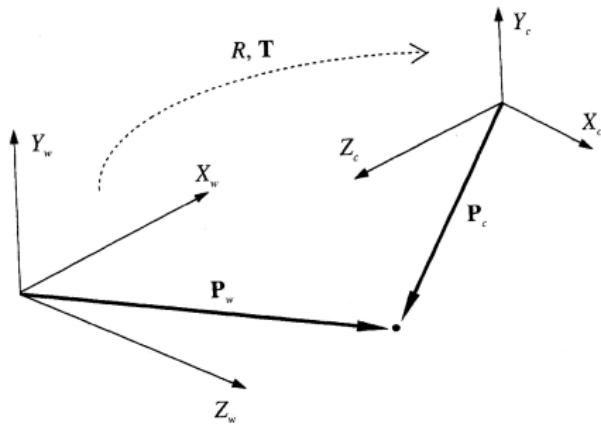


O modelo geométrico de câmera perspectiva consiste em um plano π que é o *plano da imagem* e um sistema de coordenadas com origem **O**, chamado *centro de projeção*. A distância do ponto **O** ao plano π , indicada por f , é a *distância focal*.

Nesse sistema, dado um ponto $P = [X, Y, Z]^T$ e sua imagem $p = [x, y, z]^t$ no plano π , são imediatas as equações fundamentais

$$x = f \frac{X}{Z} \quad \text{e} \quad y = f \frac{Y}{Z}$$

Geometria epipolar



Geralmente, um ponto P_c no sistema de coordenadas da câmera não coincide com suas coordenadas no P_w do mundo. A imagem ilustra essa diferença.

Então, uma matriz de rotação R e um vetor de translação T , chamados parâmetros extrínsecos, são usados para estabelecer a seguinte relação

$$P_c = R(P_w - T)$$

Geometria epipolar

Além dos parâmetros extrínsecos, parâmetros *intrínsecos* são usados para obter coordenadas, em pixels $p_{im} = [x_{im}, y_{im}]$, da imagem a partir de suas coordenadas $P_c = [X_c, Y_c, Z_c]$ no sistema de referência da câmera.

$$x_{im} = o_x - \frac{f}{s_x} \frac{X_c}{Z_c} \quad \text{e} \quad y_{im} = o_y - \frac{f}{s_y} \frac{Y_c}{Z_c}$$

Na forma matricial, os parâmetros intrínsecos definem a matriz M_{int} , dada por

$$M_{int} = \begin{bmatrix} \frac{-f}{s_x} & 0 & o_x \\ 0 & \frac{-f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix}.$$

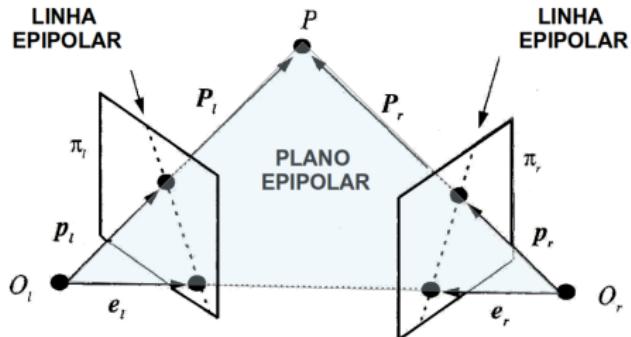
Geometria epipolar

Fixado um ponto na imagem da direita, se a busca pelo correspondente na imagem esquerda considerar todos os pontos, essa busca seria altamente complexa.

A **geometria epipolar** permite que, para cada ponto na imagem da direita, a busca pelo correspondente na imagem da esquerda seja restrita a uma linha e, portanto, tornando a busca mais eficiente.

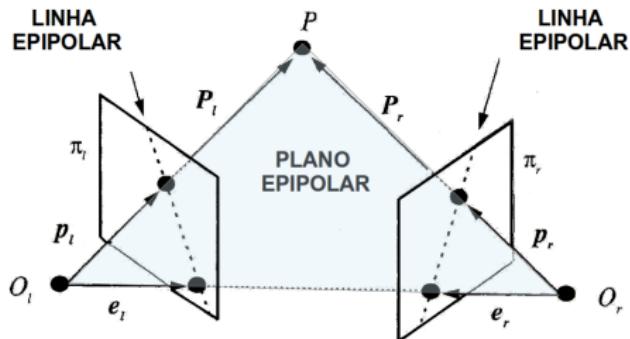


Geometria epipolar



- O_l e O_r são os centros de projeção das câmeras C_l e C_r ;
- π_l e π_r são os planos de imagem;
- f_l e f_r são as distâncias focais;
- \mathbf{P}_l e \mathbf{P}_r são coordenadas de P nos sistemas C_l e C_r
- \mathbf{p}_l e \mathbf{p}_r são projeções de \mathbf{P}_l e \mathbf{P}_r ;

Geometria epipolar

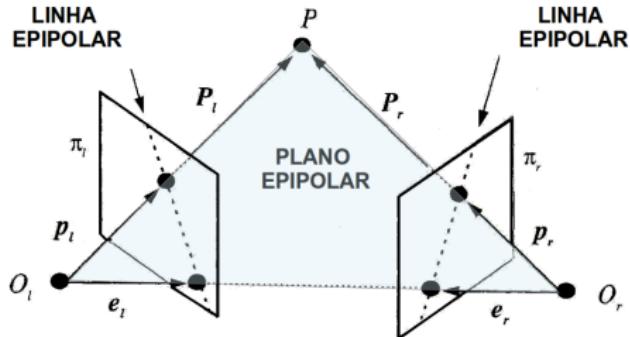


A linha definida pelos centros de projeção O_l e O_r intercepta os planos π_l e π_r nos pontos e_l e e_r denominados **epipolos**

Em cada imagem, a linha definida pelo epipolo (e_l ou e_r) com a imagem do ponto (p_l ou p_r) é chamada **linha epipolar**

O plano definido por P e os centros de projeção, chamado **plano epipolar**, intercepta os planos de imagem nas linhas epipolares.

Geometria epipolar



Uma rotação R e uma translação T , associa o ponto \mathbf{P}_r ao ponto \mathbf{P}_l , fazendo

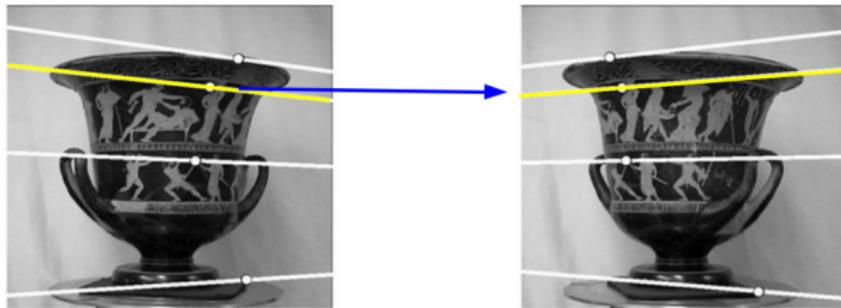
$$\mathbf{P}_r = R(\mathbf{P}_l - T)$$

Além disso, temos que

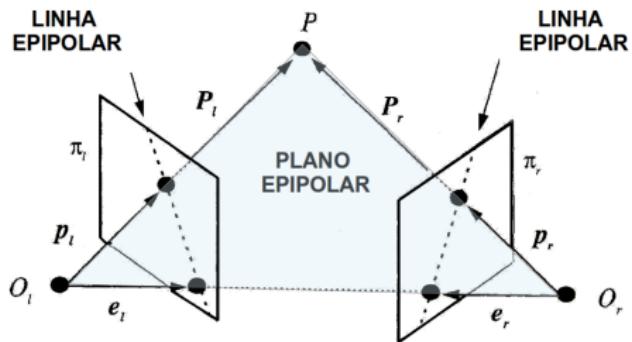
$$p_r = \frac{f_r \cdot \mathbf{P}_r}{Z_r} \quad \text{e} \quad p_l = \frac{f_l \cdot \mathbf{P}_l}{Z_l}$$

Geometria epipolar

A estimativa da geometria epipolar trata de estimar uma matriz **essencial E** e uma matriz **fundamental F** que permitem a associação de linhas epipolares na imagem da direita com sua correspondente na imagem da esquerda.



Geometria epipolar



Começamos observando que são coplanares \mathbf{P}_l , $T = (O_r - O_l)$ e $\mathbf{P}_l - T$. Então

$$(\mathbf{P}_l - T)^T \cdot (\mathbf{T} \times \mathbf{P}_l) = 0$$

Geometria epipolar

De $(\mathbf{P}_l - T)^T \cdot (T \times \mathbf{P}_l) = 0$

e observando que $\mathbf{P}_r = R(\mathbf{P}_l - T) \Rightarrow \mathbf{P}_l - T = R^T \mathbf{P}_r$,

temos então que $(R^T \mathbf{P}_r)^T \cdot (T \times \mathbf{P}_l) = 0$

Ou seja, $\mathbf{P}_r^T \cdot R \cdot T \times \mathbf{P}_l = 0$.

Isso implica $\mathbf{P}_r^T \cdot R \cdot S \cdot \mathbf{P}_l = 0$, fazendo $S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}$.

Finalmente, tomamos $E = R \cdot S$ e temos $\mathbf{P}_r^T \cdot E \cdot \mathbf{P}_l = 0$,

A matriz E é chamada **matriz essencial** e estabelece uma relação entre as restrições epipolares e os parâmetros extrínsecos.

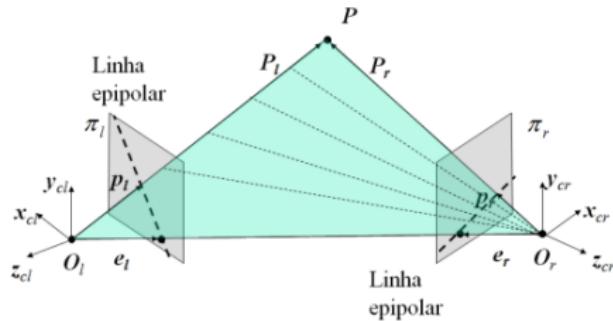
Geometria epipolar

De $\mathbf{P}_r^T \cdot E \cdot \mathbf{P}_l = 0$,

e observando que $p_r = \frac{f_r \cdot \mathbf{P}_r}{Z_r}$ e que $p_l = \frac{f_l \cdot \mathbf{P}_l}{Z_l}$

temos que $\mathbf{P}_r^T \cdot E \cdot \mathbf{P}_l = 0 \Rightarrow \frac{Z_r \cdot p_r}{f_r} \cdot E \cdot \frac{Z_l \cdot p_l}{f_l} = 0 \Rightarrow p_r^T \cdot E \cdot p_l = 0$,

Então, a matriz E , associa o ponto p_l , na imagem esquerda, à linha projetiva $E \cdot p_l$ que passa por p_r e pelo epipolo e_r .



Geometria epipolar

Deve-se observar que p_l e p_r não são dados em pixels, mas nos sistemas de coordenadas das câmeras esquerda e direita, respectivamente.

O mapeamento entre pontos p no sistema de coordenadas da câmera e \bar{p} , dado em pixel, é obtido por $\bar{p} = M.p$, onde

$$M = \begin{bmatrix} -f & 0 & o_x \\ \frac{-f}{s_x} & \frac{-f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix}.$$

Então, $p_r = M_r^{-1} \cdot \bar{p}_r$ e, analogamente, $p_l = M_l^{-1} \cdot \bar{p}_l$.

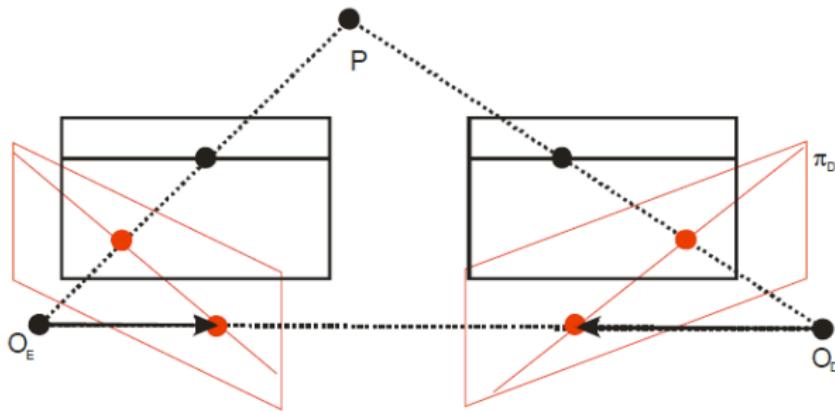
De $p_r^T \cdot E \cdot p_l = 0$, temos que $\bar{p}_r^T \cdot M_r^{-T} \cdot E \cdot M_l^{-1} \cdot \bar{p}_l = 0$.

Fazendo, $F = M_r^{-T} \cdot E \cdot M_l^{-1}$, temos que $\bar{p}_r^T \cdot F \cdot \bar{p}_l = 0$.

A matriz F é chamada **matriz fundamental** e associa o ponto \bar{p}_l , em pixel, na imagem esquerda à linha projetiva $F \cdot \bar{p}_r$, que passa por \bar{p}_r , na imagem direita.

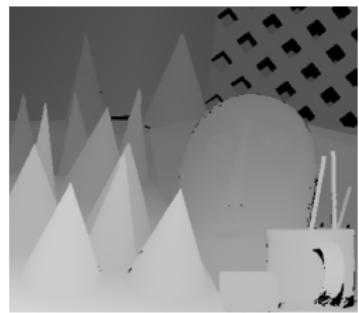
Geometria epipolar

Uma vez obtidas as linhas epipolares e suas correspondentes nas duas imagens, o processo de *retificação* transforma as imagens de forma a ter linhas epipolares paralelas ao eixo x.



Geometria epipolar

A matriz fundamental F permite, portanto, um recurso eficiente para se restringir o espaço de busca na tarefa de correspondência.



Dado um pixel $[x, y]$ na imagem esquerda, os candidatos a correspondentes na imagem direita são da forma $[x + h, y]$.

Geometria epipolar

Entretanto, existe o inconveniente de que a construção de F depende dos parâmetros extrínsecos R e T , e dos parâmetros intrínsecos f_r, s_x, s_y, o_x e o_y da câmera da direita, bem como da câmera da esquerda.

Alternativamente, a matriz F pode ser estimada a partir do conhecimento de, pelo menos, 8 pontos correspondentes entre as imagens da direita e da esquerda.

Essa estratégia, conhecida como **algoritmo dos oito pontos**, modela o problema como solução não-trivial de um sistema linear homogêneo.

Geometria epipolar

A relação $\bar{p}_r^T \cdot F \cdot \bar{p}_l = 0$ é uma equação linear homogênea de 9 incógnitas que são as entradas de F , desconhecidas.

Então, conhecidos 8 pares de pontos correspondentes, teremos 8 dessas equações que levam a um sistema linear homogêneo de 9 incógnitas e 8 equações.

A solução não-trivial desse sistema é única, a menos de escala e sinal.

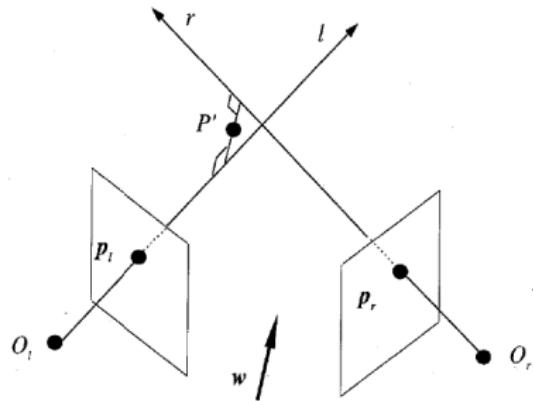
Considerando a presença de ruído na estimativa dos pares correspondentes, mais de 8 pontos podem ser usados de forma a ter um sistema sobredeterminado cuja solução pode ser encontrada usando decomposição em valores singulares.

Reconstrução 3D

A reconstrução 3D permite recuperar a exata coordenada de um ponto \mathbf{P} no sistema de coordenadas do mundo, desde que se conheça:

- Sua projeção \mathbf{p}_l e \mathbf{p}_r num par estereo;
- Parâmetros extrínsecos;
- Parâmetros intrínsecos.

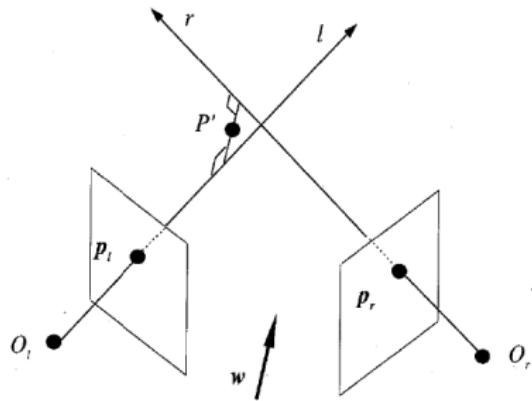
Reconstrução 3D



O ponto \mathbf{P} , é obtido como intersecção dos raios $\overline{\mathbf{O}_l \mathbf{p}_l}$ e $\overline{\mathbf{O}_r \mathbf{p}_r}$.

Então, conhecendo \mathbf{O}_l , \mathbf{O}_r , \mathbf{p}_l , \mathbf{p}_r , é possível computar as coordenadas de \mathbf{P} .

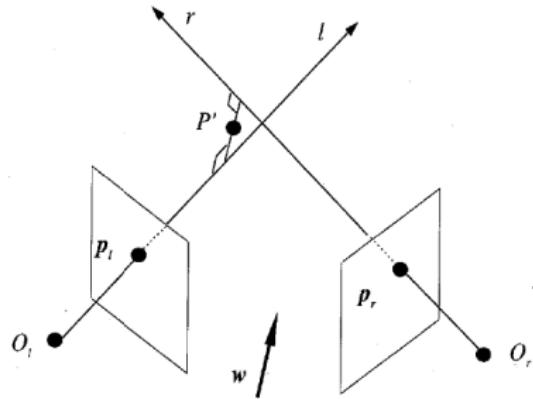
Reconstrução 3D



Entretanto, como os parâmetros extrínsecos e intrínsecos, bem como a localização dos pontos na imagem são obtidos por aproximação, essa intersecção pode não existir.

Então, um ponto \mathbf{P}' é estimado como o ponto mais próximo dos aos raios.

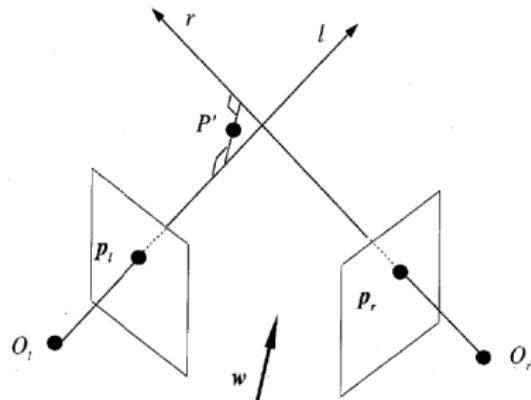
Reconstrução 3D



Consideremos:

- $\mathbf{l} = \mathbf{a}\mathbf{p}_l$, o raio com origem \mathbf{O}_l passando por \mathbf{p}_l
- $\mathbf{r} = \mathbf{T} + \mathbf{b}\mathbf{R}^t\mathbf{p}_r$, o raio com origem \mathbf{O}_r passando por \mathbf{p}_r
- $\mathbf{w} = (\mathbf{p}_l \times \mathbf{R}^t\mathbf{p}_r)$, que é ortogonal aos raios \mathbf{l} e \mathbf{r}

Reconstrução 3D

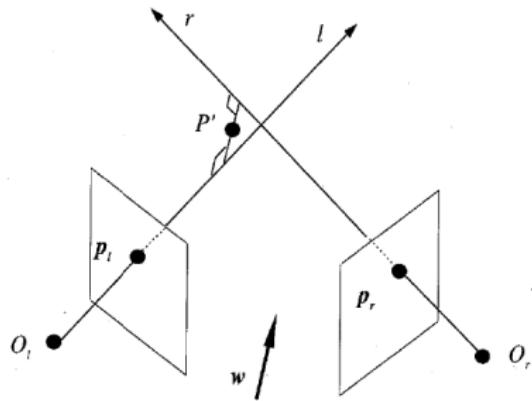


Consideremos:

- $\mathbf{l} = \mathbf{a}\mathbf{p}_l$, o raio com origem \mathbf{O}_l passando por \mathbf{p}_l
- $\mathbf{r} = \mathbf{T} + \mathbf{b}\mathbf{R}^t\mathbf{p}_r$, o raio com origem \mathbf{O}_r passando por \mathbf{p}_r
- $\mathbf{w} = (\mathbf{p}_l \times \mathbf{R}^t\mathbf{p}_r)$, que é ortogonal aos raios \mathbf{l} e \mathbf{r}

Então, \mathbf{P}' será o ponto médio do segmento paralelo a \mathbf{w} , que une \mathbf{l} e \mathbf{r} .

Reconstrução 3D



A equação $a\mathbf{l} + c\mathbf{w} - b\mathbf{r} = \mathbf{T}$, diz que, percorrendo \mathbf{l} por um fator a , de \mathbf{w} por um fator c e retornando por \mathbf{r} por um fator b , é equivalente a percorrer o vetor \mathbf{T} .

Isso leva a um sistema de equações nas incógnitas a, b, c .

Reconstrução 3D

A equação $a\mathbf{l} + c\mathbf{w} - b\mathbf{r} = \mathbf{T}$, leva ao sistema

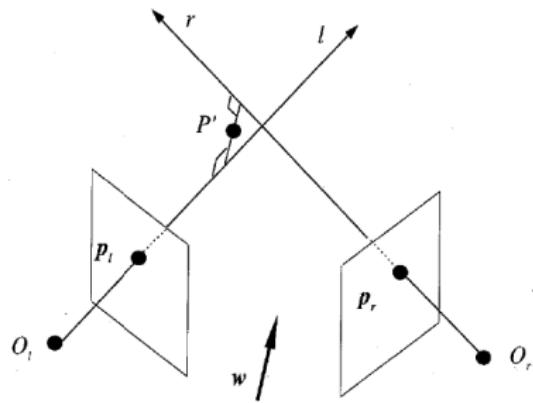
$$a \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix} - b \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} + c \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}$$

ou, na forma matricial

$$\begin{pmatrix} l_x & r_x & w_x \\ l_y & r_y & w_y \\ l_z & r_z & w_z \end{pmatrix} \begin{pmatrix} a \\ -b \\ c \end{pmatrix} = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}$$

Note que o sistema tem solução única sempre que os raios \mathbf{r} e \mathbf{l} não são paralelos.

Reconstrução 3D



De posse da solução a_0, b_0, c_0 , para a equação $al + cw - br = \mathbf{T}$, obtemos os pontos $\mathbf{Q}_l = a_0 p_l$ e $\mathbf{Q}_r = T + b_0 R^t p_r$, que são os pontos mais próximos entre os raios l e r . O ponto \mathbf{P}' é então tomado como

$$\mathbf{P}' = \frac{\mathbf{Q}_l + \mathbf{Q}_r}{2}$$

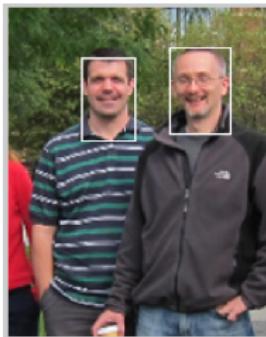
Detecção e Reconhecimento de Faces

Entre as tarefas mais desafiadoras que um computador pode realizar estão as tarefas de **detecção** e **reconhecimento** de objetos em uma cena.

- **Detecção:** Identificar a presença de um objeto e sua localização na cena;
- **Reconhecimento:** Dado uma instância do objeto, classifica-lo como um indivíduo dentro da classe de objetos de mesma natureza.

Detecção e Reconhecimento de Faces

Entre os algoritmos de detecção e reconhecimento mais populares estão os a relacionados **faces humanas**.



Também são populares algoritmos de reconhecimento de pedestres.



Detecção e Reconhecimento

As técnicas de detecção de faces podem ser classificadas como baseadas em:

- **Características:** Identificam regiões como pele, olhos, nariz, boca e testam a consistência;
- **Modelos:** Uma variedade de poses e expressões (máscaras) são consideradas e testadas contra cada posição;
- **Aparência:** Uma janela retangular com máscara simples identifica candidatos a serem testados numa etapa de refinamento.

Entre os mais conhecidos, o detector proposto por **Viola e Jones (2004)** é amplamente usado.

Nele, uma série de classificadores de aparência simples são combinados numa técnica chamada **Boosting**.

Detecção e Reconhecimento

Num exemplo de detecção de faces por características, usaremos tons de pele para isolar candidatos.

Alternativamente ao espaço de cores RGB, a codificação **YCbCr** especifica as cores independentemente da intensidade.

Enquanto que **Cb,Cr** guarda a cromaticidade das cores independentemente da intensidade, **Y** guarda a luminância. A conversão se dá pelas equações

$$Y = +0.257R + 0.504G + 0.098B$$

$$Cb = -0.148R - 0.291G + 0.439B + 0.5$$

$$Cr = +0.439R - 0.368G - 0.071B + 0.5$$

Experimentalmente, verifica-se que os tons de pele encontram-se em

$$10 \leq Cr \leq 45$$

Detecção de faces

Então, o primeiro passo do nosso algoritmo para detecção de faces deve calcular **Cr**

```
function Cr=facedetect(I)
[l c n]=size(I);
I=double(I);
Cr=0.439*I(:,:,1) - 0.368*I(:,:,2) - 0.071*I(:,:,3) + 0.5;
```

Depois, identificar os pixels, correspondentes a cor de pele

```
S=zeros(l,c);
[L, C] = find(10<Cr & Cr<45);
for i=1:size(L)
    S(L(i),C(i))=1;
end
```

Detecção de faces

Cada agrupamento de pixels associados a cor de pele deve receber um índice.

```
r=2;  
BB=[ ];  
for i=1:l  
    for j=1:c  
        x1=j; y1=i; x2=j; y2=i;  
        if (S(i,j)==1)  
            neighbors(i,j,r);  
            r=r+1;  
            BB=[BB; [x1 y1 x2 y2]];  
        end  
    end  
end
```

A função **neighbors** é recursiva de forma que, dado um pixel **(i,j)**, todos os pixels da mesma componente conexa recebem label **r**.

Detecção de faces

```
function [S,W]=neighbors(S,W,i,j,r)
[l c]=size(s);
if (s(i,j)==1)
    s(i,j)=r;
    [S,W]=neighbors(S,W,min(i+1,l),j,r);
    [S,W]=neighbors(S,W,i,min(j+1,c),r);
    [S,W]=neighbors(S,W,max(1,i-1),j,r);
    [S,W]=neighbors(S,W,i,max(1,j-1),r);
    W=[min(j,W(1)) min(i,W(2)) max(j,W(3)) max(i,W(4))];
end
```

Além de etiquetar as componentes conexas, a função guarda o **bounding box** $[x_1, x_2, y_1, y_2]$ de cada componente.

Detecção de faces

Finalmente, nosso código fica:

```
function S=facedetect(I)
[l c n]=size(I); I=double(I);
Cr=0.439*I(:,:,1) - 0.368*I(:,:,2) - 0.071*I(:,:,3) + 0.5;

S=zeros(l,c);
[L,C] = find(10<Cr & Cr<45);
for i=1:size(L)
    S(L(i),C(i))=1;
end

r=2;
BB=[];
for i=1:l; for j=1:c
    W=[j i j i];
    if (S(i,j)==1)
        [S,W]=neighbors(S,W,i,j,r);
        r=r+1;
        BB=[BB; W];
    end
end; end

w=BB(:,3)-BB(:,1);
h=BB(:,4)-BB(:,2);

for i=1:size(BB,1)
    if (h(i)>20 & w(i)>20)
        if (h(i)/w(i)<2 & h(i)/w(i)>0.8)
            I(BB(i,2),BB(i,1):BB(i,3),:)=255;
            I(BB(i,4),BB(i,1):BB(i,3),:)=255;
            I(BB(i,2):BB(i,4),BB(i,1),:)=255;
            I(BB(i,2):BB(i,4),BB(i,3),:)=255;
        end
    end
end

I=uint8(I);
imshow(I);
```

```
function [S,W]=neighbors(S,W,i,j,r)
[l c]=size(S);

if (S(i,j)==1)
    S(:,j)=r;
    [S,W]=neighbors(S,W,min(i+1,l),j,r);
    [S,W]=neighbors(S,W,i,min(j+1,c),r);
    [S,W]=neighbors(S,W,max(i,l-1),j,r);
    [S,W]=neighbors(S,W,i,max(1,j-1),r);

    W=[min(j,W(1)) min(i,W(2)) max(j,W(3)) max(i,W(4))];

end
```

Reconhecimento de faces

Para o Reconhecimento de Faces uma técnica comum, chamada **Eigenfaces**, usa decomposição em valores singulares.

Dado um conjunto de K imagens de dimensão $M \times N$, cada uma sendo imagem de uma face,

- Cada imagem se torna um vetor de $D = M \cdot N$ posições;
- O conjunto é separado em imagens de **treino** e imagens de **teste**;
- O conjunto de treino define uma matriz M , onde cada coluna é uma imagem de treino
- Obtem a imagem média $med = mean(M, 2)$;
- Todas as colunas de M são subtraídas da média $M(:, i) = M(:, i) - med$;
- Decompõe $M = USV$ usando de valores singulares, $[USV] = svd(M)$;
- Cada coluna de U é chamada eigenface, ou face principal;
- Projetamos cada face nas primeiras h componentes, $F = U(:, 1 : h)' * M$ de forma que cada face é combinação linear de h eigenfaces;
- Dada uma face de teste t , escrevemos $t = U(:, 1 : h)' * (t - med)$ e comparamos com as faces de treino;

Reconhecimento de faces

Considere o conjunto de faces de treino:



E o conjunto de Imagens de teste:



Reconhecimento de faces

Em nosso exemplo, temos

- 60 imagens 112×92
- 20 indivíduos, 3 imagens de cada um
- Cada imagem gera um vetor de 10.304 posições

Então construimos a matriz M , 60×10304 , onde cada coluna corresponde a uma imagem. A média das colunas de A resulta na face média.



Reconhecimento de faces

Removendo a média e efetuando a decomposição SVD, obtemos

$$M = USV^t \quad (4)$$

A base U , 10304×10304 , tem, em cada coluna uma imagem chamada de “eigenface”.

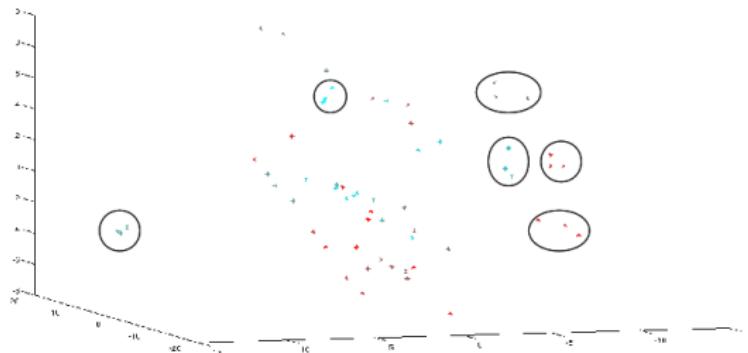


Cada imagem original é uma composição de “eigenfaces”. Na prática, apenas 60 “eigenfaces” são suficientes para compor todas as imagens originais.

Reconhecimento de faces

Na prática, apenas algumas poucas “eigenfaces” são suficientes para compor todas as imagens originais.

Finalmente, a comparação entre duas faces para reconhecimento é feita pelo vizinho mais próximo no espaço das eigenfaces.



Reconhecimento de faces

```
function [U F med]=trainface(a,b)

M=[];
for i=1:60
    if (mod(i,3)==a || mod(i,3)==b)
        imagem=sprintf('img/%i.png',i);
        A=imread(imagem);
        A = rgb2gray(A);
        A = double(A)/255;
        [L C]=size(A);
        e=reshape(A,L*C,1);
        M=[M e];
    end
end

k=size(M,2);

med=mean(M,2);

for i=1:k
    M(:,i) = M(:,i) - med;
end

[U S V] = svd(M);

F=U(:,1:k)'*M;
```

```
function C=testface(F,U,med,c)

CC = zeros(20,20);
k=size(F,2);

for i=1:60
    if (mod(i,3)==c)
        c1 = ceil(i/3)
        imagem=sprintf('img/%i.png',i);
        A=double(rgb2gray(imread(imagem)))/255;
        [L C]=size(A);
        e=reshape(A,L*C,1);
        f = U(:,1:k)'*(e-med);

        for j=1:k
            tmp(j) = norm(F(:,j)-f,2);
        end
        [w c2]=min(tmp);
        c2=ceil(c2/2);
        CC(c1,c2)=CC(c1,c2)+1;
    end
end

sum(diag(CC))/sum(sum(CC))

imshow(1-CC);
```