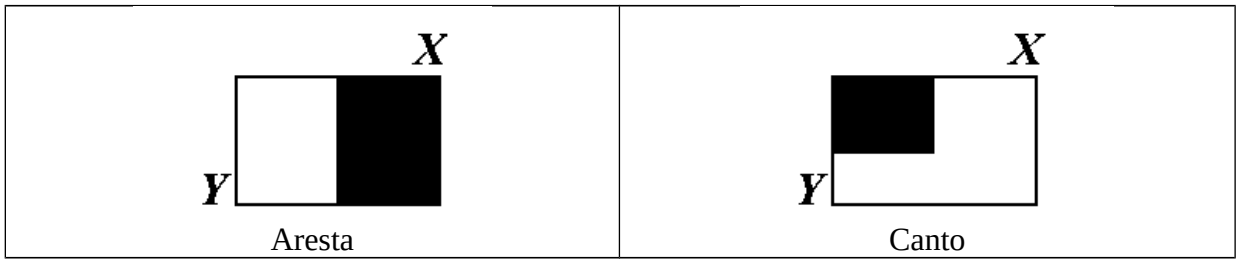
Detecção de cantos (corners) em imagens em níveis de cinza

Diferenças entre uma aresta e uma esquina:

- Uma aresta é um local da imagem onde a variação local (derivada parcial) de f(x,y) é alta numa certa direção e baixa na direção ortogonal.
- Uma esquina é um local da imagem onde as variações locais (derivadas parciais) de f(x,y) são altas nas direções X e Y.

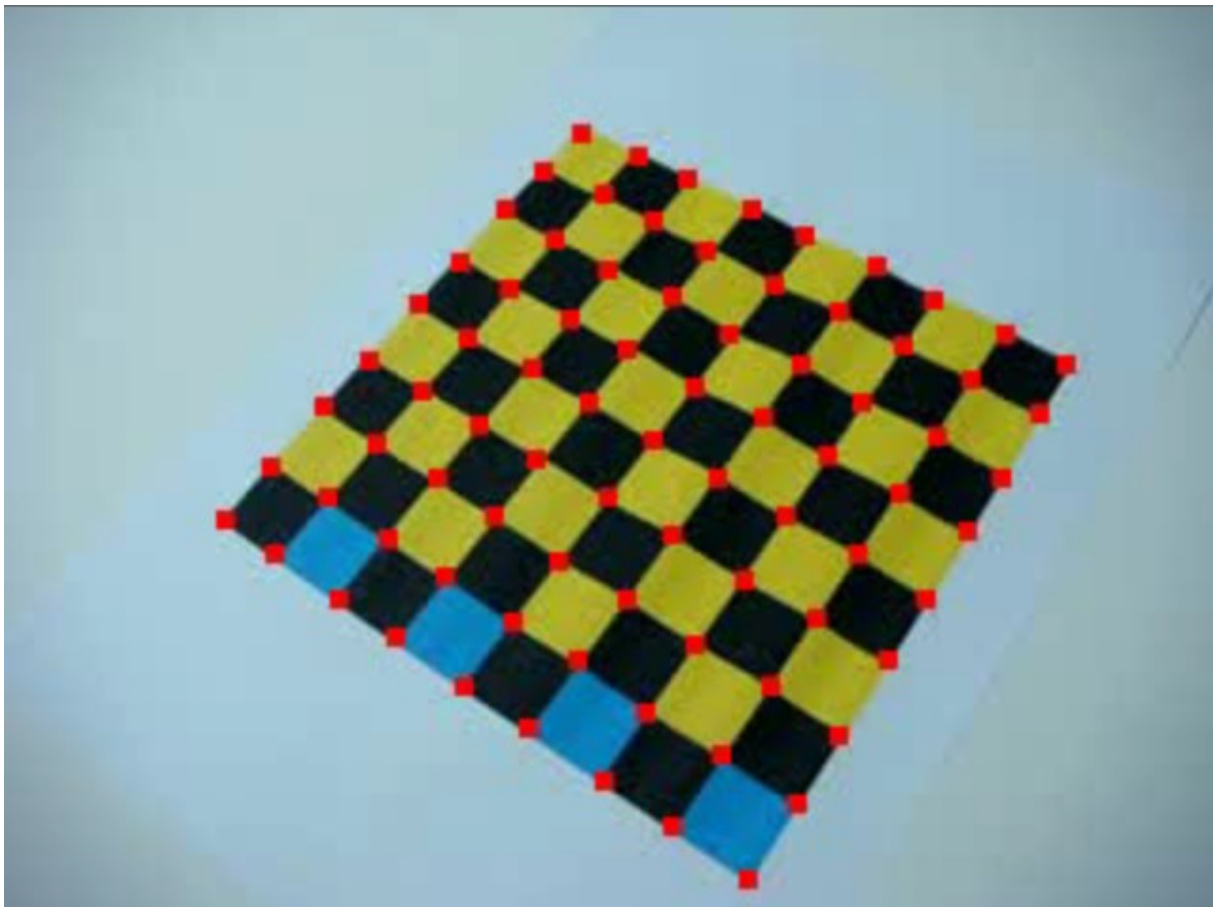
Esquinas são usadas em análise de forma e de movimento, estereoscopia, busca de imagem em banco de dados.

- Movimento é ambíguo numa aresta, mas não-ambíguo numa esquina.
- Formas podem ser reconstruídas aproximadamente a partir das esquinas.



Aresta | Canto

Exemplo de corner Harris de OpenCV



```
#include <cekeikon.h>

Mat_<FLT> cornerHarris(Mat_<COR> a)
{ vector< Mat_<GRY> > v;
  split(a,v);
  vector< Mat_<FLT> > w(3);
// void cornerHarris(InputArray src, OutputArray dst, int blockSize,
//                   int ksize, double k, int borderType=BORDER_DEFAULT )
  for (unsigned i=0; i<w.size(); i++) {
    cornerHarris(v[i], w[i], 3, 3, 0.04);
    w[i]=max(w[i], 0.0);
  }
  Mat_<FLT> m=max(w[0],w[1]);
  m=max(m,w[2]);
  m=30.0*m;
  return m;
}

int main(int argc, char** argv)
{ if (argc!=4) erro("Harris entrada.ppm saida.ppm n");
  Mat_<COR> a; le(a,argv[1]);
  int n; convArg(n,argv[3]);
  Mat_<FLT> m=cornerHarris(a);

  vector<Point> vp=kmax(m,n,10.0);
  Mat_<COR> b=a.clone();
  for (unsigned i=0; i<vp.size(); i++) {
    Point p=vp[i];
```

```
      ponto(b,p.y,p.x,COR(0,0,255),5);
    }
    imp(b,argv[2]);
}
```

Detector de esquina de Harris:

Considere uma janela *W* situada numa posição (*u*,*v*) da imagem *I*. Se a janela *W* se deslocar um pouco na direção (*x*,*y*), o nível de cinza dentro da janela sofrerá alteração. Neste caso,

1) Se a janela *W* está numa região com nível de cinza constante, então deslocamento em qualquer direção resultará em pequena alteração no nível de cinza médio.
2) Se a janela *W* está numa aresta, então um deslocamento ao longo da aresta resultará numa mudança pequena. Mas um deslocamento perpendicular à aresta resultará numa mudança grande.
3) Se a janela *W* está numa esquina ou num ponto isolado, deslocamento em qualquer direção resultará em mudança grande.

Considere pegar a região em torno do pixel (*u*,*v*) da imagem *I* e deslocá-la por (*x*,*y*). A diferença quadrática ponderada *S* entre as duas regiões será dada por:

$$S(x,y) = \sum_{u,v} w(u,v)\left[I(x+u, y+v) - I(u,v)\right]^2$$

Aproximando $I(x+u, y+v)$ pela expansão de Taylor:

$$I(x+u, y+v) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y$$

obtemos:

$$S(x,y) \approx \sum_{u,v} w(u,v)\left|I_x(u,v)x + I_y(u,v)y\right|^2$$

onde I_x e I_y são as derivadas parciais de *I*. Equivalentemente:

$$S(x,y) \approx (x,y)A\begin{bmatrix} x \\ y \end{bmatrix}$$

onde *A* é a tensor de estrutura:

$$A = \sum_{u,v} w(u,v)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_xI_y \rangle \\ \langle I_xI_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

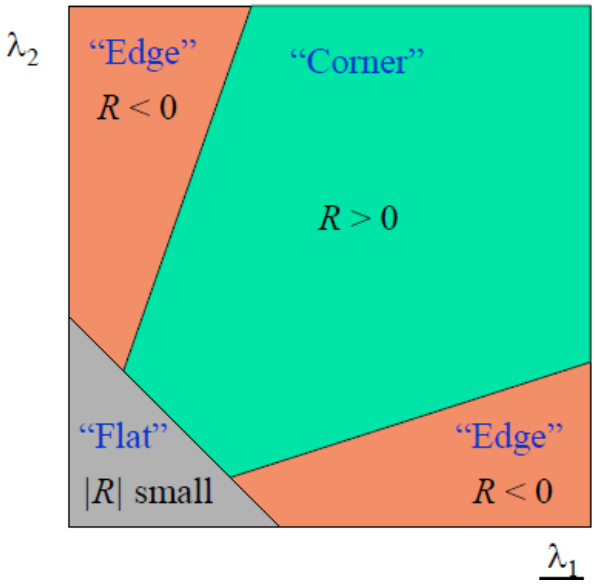
onde <.> indica calcular a média local. Se uma janela circular (como a janela gaussiana) for usada, então a resposta será isotrópica.

Os dois auto-valores de uma matriz 2*2 pode ser calculada:

$$\lambda_{\pm} = \frac{1}{2}\left[(a_{11} + a_{22}) \pm \sqrt{4a_{12}a_{21} + (a_{11} - a_{22})^2}\right]$$

Os dois auto-valores da matriz *A* indicam:

1. Se $\lambda_1 \approx 0$ e $\lambda_2 \approx 0$ então o pixel (*x*,*y*) está numa região com o nível de cinza constante.
2. Se $\lambda_1 \approx 0$ e λ_2 for valor positivo grande, então uma aresta foi encontrada.
3. Se λ_1 e λ_2 são valores positivos grandes, então uma esquina foi encontrada.



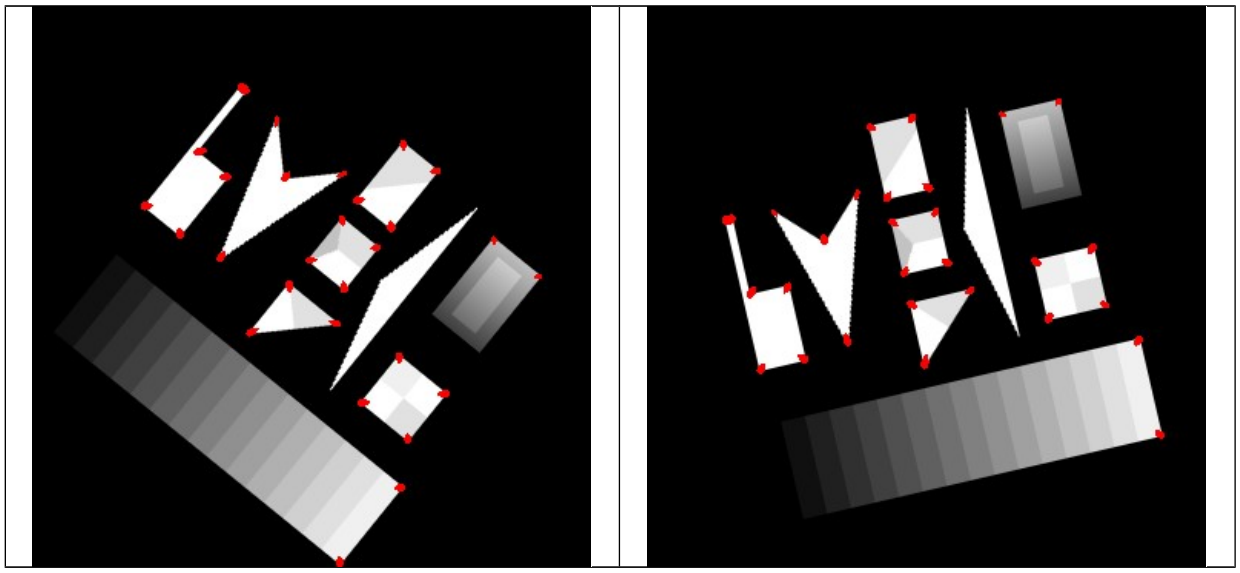
Esquina no espaço de escala:

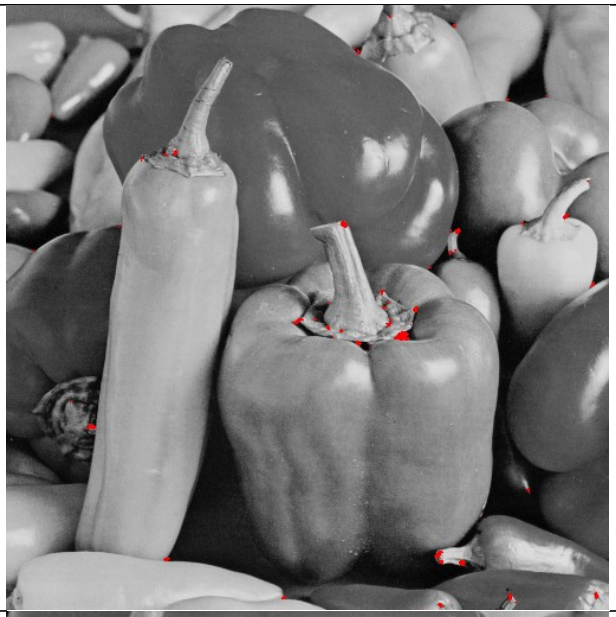
O cálculo da matriz de segundo momento *A* (também chamada de tensor de estrutura) requer a computação das derivadas parciais I_x e I_y da imagem *I*, assim como a média local dessas derivadas em vizinhança local. Como a computação das derivadas normalmente envolve um estágio de alisamento local, a definição operacional do operador de Harris requer dois parâmetros de escala:
1. Uma escala local *t* para alisar imagem no cálculo das derivadas da imagem.
2. Uma escala de integração *s* para tirar a média local das derivadas.

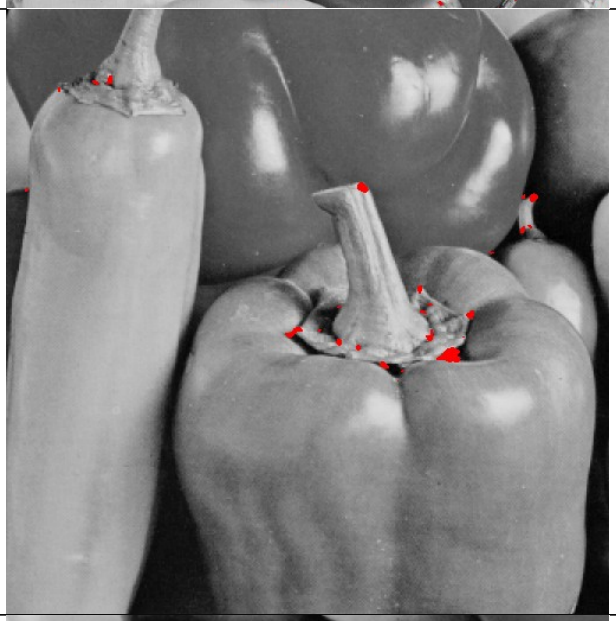
Normalmente $s = \gamma^2 t$, onde $\gamma \in \left[\sqrt{2}, 2\right]$. (ou $\gamma^2 \in [2,4]$)

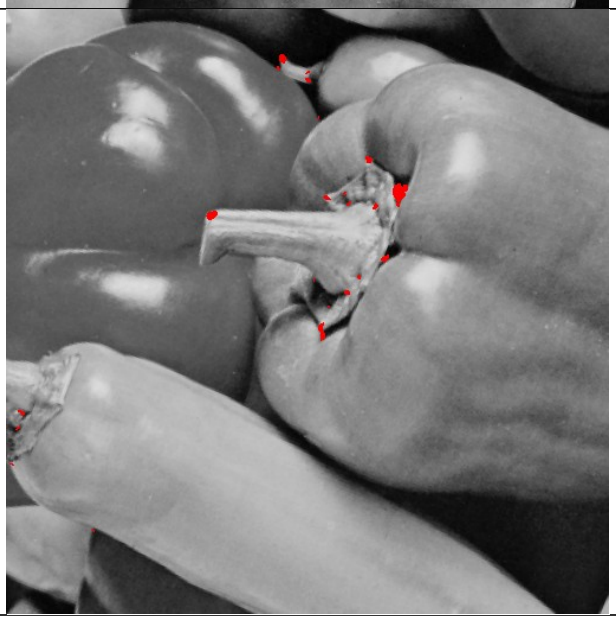
Exemplo:

```
img harris ..\susan051.pgm susan051-e.bmp 0.7 2.1 0.003
img sobrmcg ..\susan051.pgm susan051-e.bmp susan051-e3.tga n
img harris ..\susan103.pgm susan103-e.bmp 0.7 2.1 0.003
img sobrmcg ..\susan103.pgm susan103-e.bmp susan103-e3.tga n
```









Programa Harris:

```
#include <proeikon>

VETOR<double> eigenvalues22(MATRIZ<double> a)
// Retorna eigenvalues em ordem decrescente de uma matriz 2*2
// 0.0 indica ausencia de um ou dois eigenvalues
{ if (a.nl()!=2 || a.nc()!=2) erro("Erro eigenvalues22: matriz deve ser 2*2");
  VETOR<double> d(2,0.0);
  double rdelta=4*a(0,1)*a(1,0)+elev2(a(0,0)-a(1,1));
  if (rdelta==0.0) {
    d(0)=d(1)=0.5*(a(0,0)+a(1,1));
  } else if (rdelta>0.0) {
    double delta=sqrt(rdelta);
    d(0)=0.5*((a(0,0)+a(1,1))+delta);
    d(1)=0.5*((a(0,0)+a(1,1))-delta);
  }
  return d;
}

void harris(int argc, char** argv)
{ if (argc!=6) {
    printf("Harris: Detector de canto Harris\n");
    printf("Harris ent.tga sai.bmp uderivada uintegral limiar\n");
    printf("  Exemplo: harris ent.tga sai.bmp 1 3 0.1\n");
    printf("  uintegral e' 2 a 4 vezes maior que uderivada\n");
    erro("Erro: numero de parametros");
  }

  IMGFLT a; le(a,argv[1]);

  double uderivada;
  if (sscanf(argv[3],"%lf",&uderivada)!=1) erro("Erro: Leitura uderivada");
  if (uderivada<=0.0) erro("Erro: uderivada<=0.0");

  double uintegral;
  if (sscanf(argv[4],"%lf",&uintegral)!=1) erro("Erro: Leitura uintegral");
  if (uintegral<=0.0) erro("Erro: uintegral<=0.0");
  if (uintegral<2*uderivada || 4*uderivada<uintegral)
    erro("Erro: uintegral<2*uderivada || 4*uderivada<uintegral");

  double limiar;
  if (sscanf(argv[5],"%lf",&limiar)!=1) erro("Erro: Leitura limiar");
  if (limiar<=0) erro("Erro: limiar<=0");

  a=ConvGaussHV(a,uderivada);
  IMGFLT Ix=uderivada*convolucao(a,IMGFLT(1,3,  0.5, 0.0,-0.5));
  IMGFLT Iy=uderivada*convolucao(a,IMGFLT(3,1, -0.5, 0.0, 0.5));

  IMGFLT Ix2=elev2(Ix);        IMGFLT Iy2=elev2(Iy);        IMGFLT IxIy=Ix*Iy;
  Ix2=ConvGaussHV(Ix2,uintegral); Iy2=ConvGaussHV(Iy2,uintegral);
  IxIy=ConvGaussHV(IxIy,uintegral);

  //IMGFLT m0(a.nl(),a.nc());
  IMGFLT m1(a.nl(),a.nc());
  VETOR<double> d; MATRIZ<double> v;
  for (int l=0; l<a.nl(); l++)
    for (int c=0; c<a.nc(); c++) {
      MATRIZ<double> a(2,2, Ix2(l,c), IxIy(l,c), IxIy(l,c), Iy2(l,c));
      d=eigenvalues22(a);
      //m0(l,c)=d(0); // o maior eigenvalue
      m1(l,c)=d(1); // o menor eigenvalue
    }
  IMGBIN b(a.nl(),a.nc(),branco);
  for (int i=0; i<b.n(); i++) if (m1(i)>=limiar) b(i)=preto;
  imp(b,argv[2]);
}
```

Eigenvalues and eigenvectors of 2x2 matrices

Calculating eigenvalues and eigenvectors of matrices by hand can be a daunting task. This is why homework problems deal mostly with 2x2 or 3x3 matrices. For 2x2, 3x3, and 4x4 matrices, there are complete answers to the problem. In that case, one can give explicit algebraic formulas for the solutions. For 5x5 matrices, an explicit algebraic solution can not be found any more since one would have to give formulas of the roots of a polynomial of degree 5. Below, we have the solution for the 2x2 case. Your linear algebra teacher probably doesn't want you to know them...

Let T=a+d be the trace and D=ad-bc be the determinant of the matrix

$$A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

The eigenvalues of A are

$$L_1 = T/2 + (T^2/4-D)^{1/2}$$
$$L_2 = T/2 - (T^2/4-D)^{1/2}$$

If c is not zero, then the eigenvectors are

$$\begin{vmatrix} L_1-d \\ c \end{vmatrix}, \begin{vmatrix} L_2-d \\ c \end{vmatrix}$$

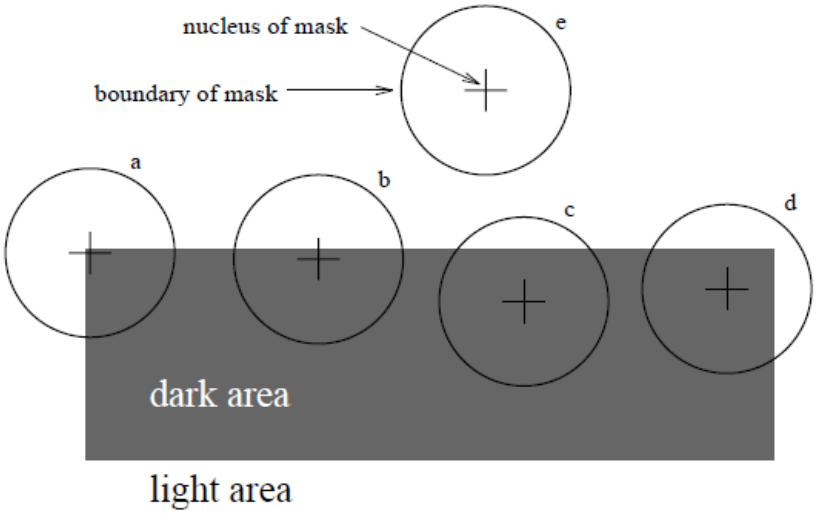
If b is not zero, then the eigenvectors are

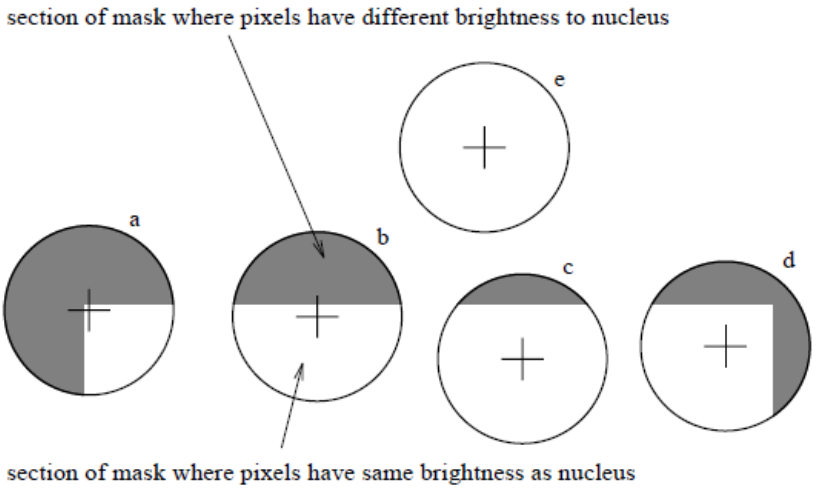
$$\begin{vmatrix} b \\ L_1-a \end{vmatrix}, \begin{vmatrix} b \\ L_2-a \end{vmatrix}$$

If both b and c are zero, then the eigenvectors are

$$\begin{vmatrix} 1 \\ 0 \end{vmatrix}, \begin{vmatrix} 0 \\ 1 \end{vmatrix}$$

SUSAN: Smallest Univalue Segment Assimilating Nucleus





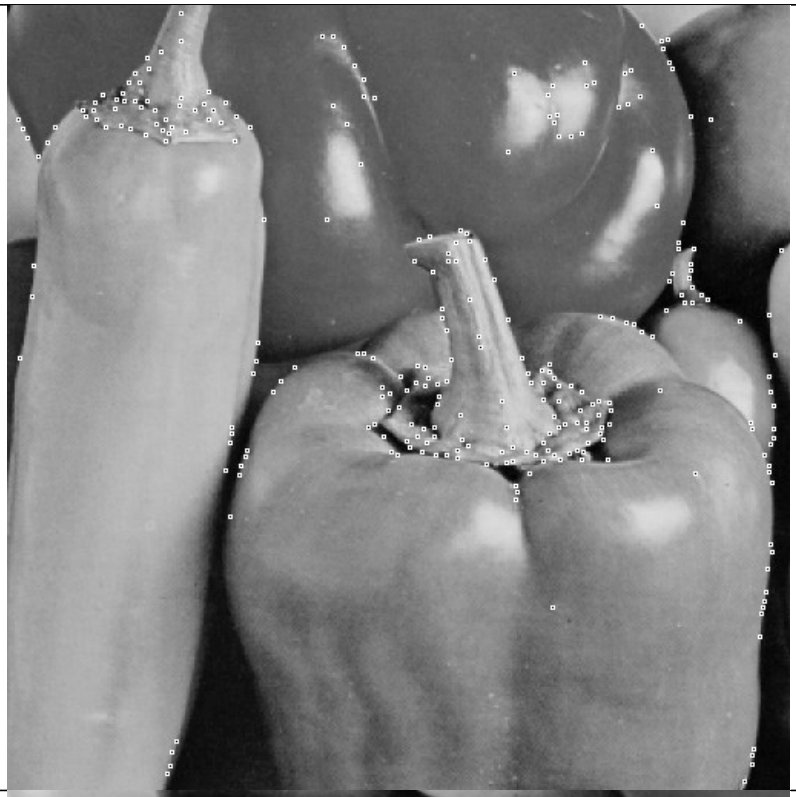
Determine uma máscara circular: tipicamente 37 pixels em torno do pixel (núcleo).
Calcule a diferença de brilho entre cada pixel na máscara e o seu núcleo:

$$c(r,r_0) = \begin{cases} 1, \text{se} \left| I(r) - I(r_0) \right| \le t \\ 0, \text{caso contrário} \end{cases}$$

Some o número de pixels com similar nível de cinza ao do núcleo

$$n(r_0) = \sum_r c(r, r_0)$$

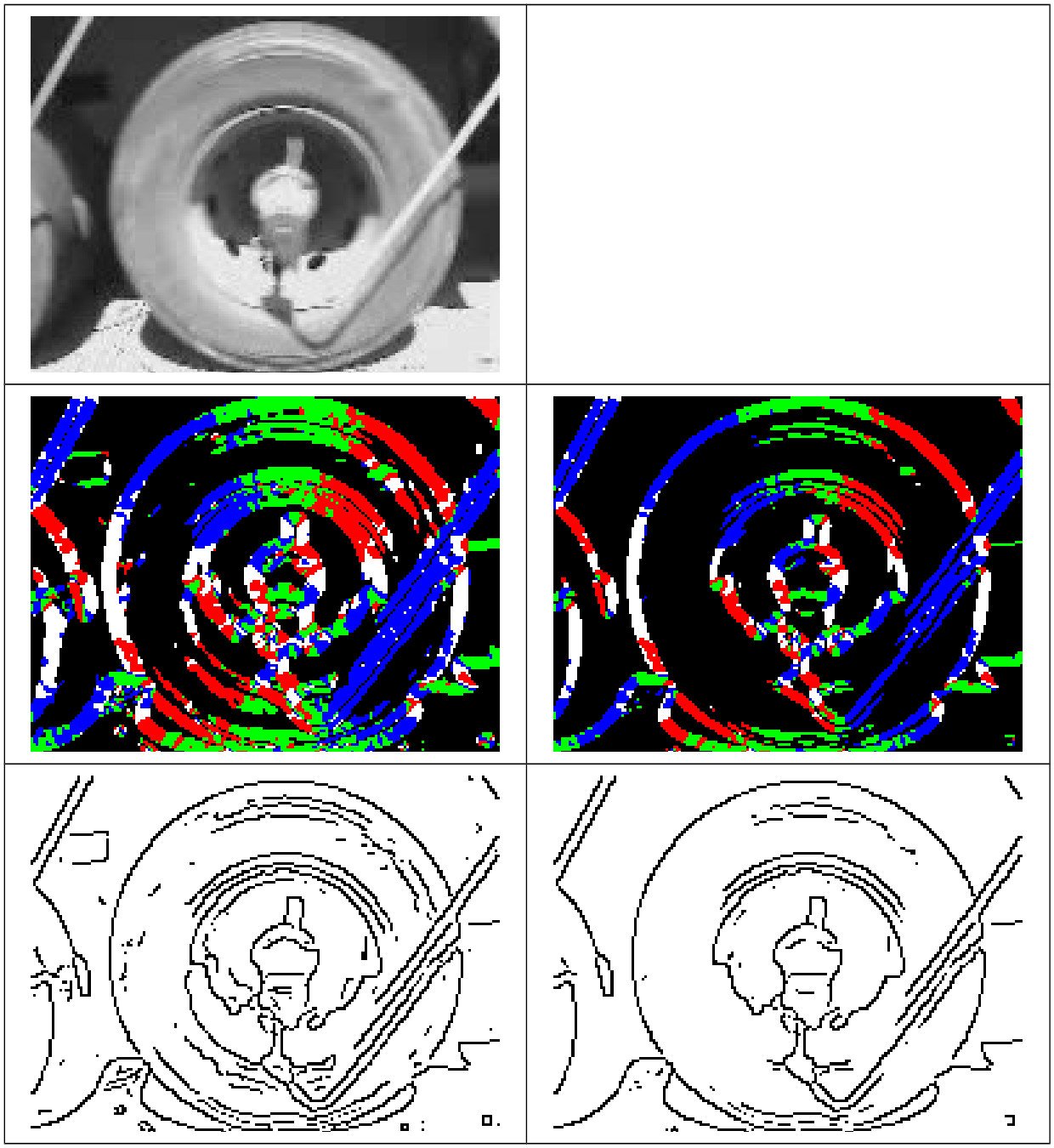
Compare *n* com *g*, o limiar geométrico que é a metade do maior valor possível de *n*.
Numa esquisa, $n(r_0)$ será menos de metade da área de máscara. E será um mínimo local.



Bibliografia:

- Harris, C. and Stephens, M. 1988. Acombined corner and edge detector. In Fourth Alvey Vision Conference, Manchester, UK, pp. 147–151.
- Smith, S.M. and Brady, J.M. 1997. SUSAN—A new approach to low level image processing. International Journal of Computer Vision, 23(1):45–78.
- Verbetes “corner detection” e “structure tensor” da Wikipedia.

Detecção de aresta de Canny:



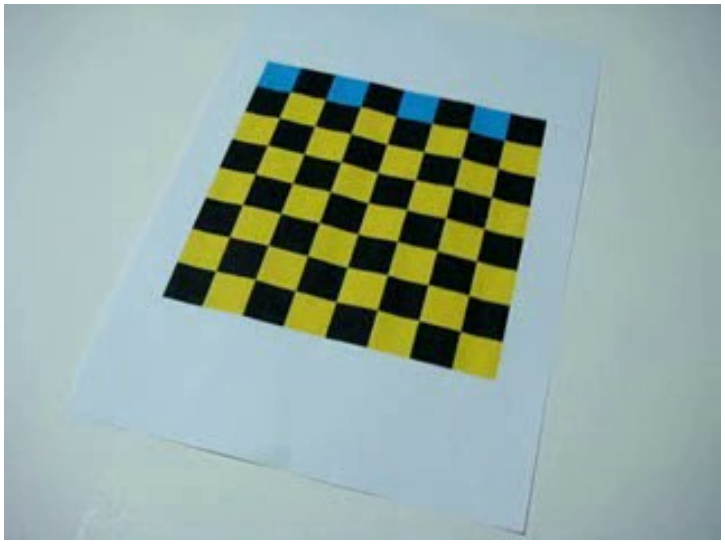
Calcula gradiente.
Limiariza o módulo de gradiente (usando limiar maior).
Calcula o ângulo de gradiente e arredonda para um número entre 0 e 3:

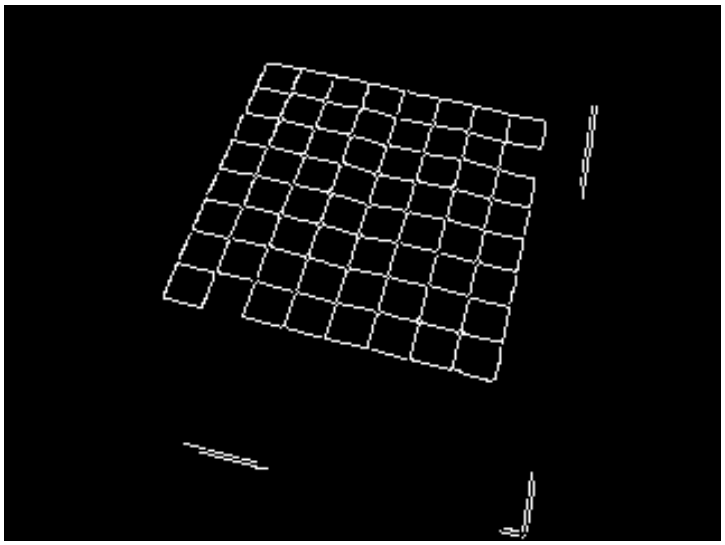
```
3   2   1
 *  *  *
  * * *
0*******0
  * * *
 *  *  *
1   2   3
```

Faz supressão de não-máximo.
Histerese - considera arestas os pixels com módulo de gradiente grande (usando limiar menor) grudados a outros pixels das arestas.

```
#include <proeikon>
#include <imgpv>

IMGBIN aresta(IMGFLT am, IMGGRY ag, double thresh)
{ IMGBIN ed(am.nl(),am.nc(),preto);
  for (int i=0; i<am.n(); i++)
    if (am(i)>thresh) ed(i)=branco;

  /*
  IMGCOR t=ed;
  for (int i=0; i<t.n(); i++)
    if (ed(i)) {
      if (ag(i)==1) t(i)=COR(255,0,0);
      else if (ag(i)==2) t(i)=COR(0,255,0);
      else if (ag(i)==3) t(i)=COR(0,0,255);
    }
  imp(t,"saida.tga");
  */

  // non maximum suppression
  for (int l=0; l<am.nl(); l++)
    for (int c=0; c<am.nc(); c++)
      if (ed(l,c)) {
        switch (ag(l,c)) {
          case 0: if (am(l,c)<am.atx(l,c-1)   || am(l,c)<am.atx(l,c+1))
                    ed(l,c)=preto; break;
          case 1: if (am(l,c)<am.atx(l+1,c-1) || am(l,c)<am.atx(l-1,c+1))
                    ed(l,c)=preto; break;
          case 2: if (am(l,c)<am.atx(l-1,c)   || am(l,c)<am.atx(l+1,c))
                    ed(l,c)=preto; break;
          case 3: if (am(l,c)<am.atx(l-1,c-1) || am(l,c)<am.atx(l+1,c+1))
                    ed(l,c)=preto; break;
          default: erro("Erro inesperado");
        }
      }
  return ed;
}

int main()
{ double thresh1=0.02;
  double thresh2=2*thresh1;
  IMGFLT a;
  le(a,"cam180.jpg");
  IMGFLT ax=gradientex(a,1.0);
  IMGFLT ay=gradientey(a,1.0);

  IMGCPX ac(ax,ay);
  IMGFLT am=abs(ac);
  IMGFLT ap=arg(ac); // fase de -pi a +pi
  for (int i=0; i<ap.n(); i++) {
    if (ap(i)<0.0) ap(i)+=M_PI; // fase de 0 a +pi
  }
  IMGGRY ag(ap.nl(),ap.nc());
  for (int i=0; i<ap.n(); i++) {
    ag(i) = arredonda(4.0*((ap(i))/M_PI));
    if (ag(i)>=4) ag(i)=0;
  }
    /*
        3   2   1
         *  *  *
          * * *
        0*******0
          * * *
         *  *  *
        1   2   3
    */

  IMGBIN ed1=aresta(am,ag,thresh1);
  imp(!ed1,"ed1.bmp");
}
```

```
#include <cekeikon.h>

Mat_<GRY> canny(Mat_<GRY> a)
{ Mat_<GRY> b;
  Canny(a, b, 160, 70);
  return b;
}

int main(int argc, char** argv)
{ if (argc!=3) erro("Canny entrada.ppm saida.ppm");
  Mat_<GRY> a; le(a,argv[1]);
  Mat_<GRY> m=canny(a);
  imp(m,argv[2]);
}
```





```
#include <proeikon>
#include <imgpv>

void canny(int argc, char** argv)
{ if (argc!=6) {
    printf("Canny ent.pgm sai.pgm thresh1 thres2 aperture\n");
    printf("  Normalmente, 2*thresh1 = thresh2, aperture=3\n");
    printf("  Ex: canny ent.pgm sai.pgm 85 170 3\n");
    erro("Erro: Numero de argumentos invalido");
  }

  IMGGRY a; pvLe(a,argv[1]);

  double thresh1;
  if (sscanf(argv[3],"%lf",&thresh1)!=1) erro("Erro: Leitura thresh1");
  double thresh2;
  if (sscanf(argv[4],"%lf",&thresh2)!=1) erro("Erro: Leitura thresh2");
  int aperture;
  if (sscanf(argv[5],"%d",&aperture)!=1) erro("Erro: Leitura aperture");

  IplImage* img=IMGGRY2pv(a);
  cvCanny( img, img, thresh1, thresh2, aperture);
  pv2IMGGRY(img,a);
  cvReleaseImage(&img);

  pvImp(a,argv[2]);
}
```