Apéndice B.

Pseudocódigos de las funciones "brightness", "user_cornerq", "user_cornerdraw".

```
// estructura de la esquina
typedef  struct {int x,y,info, dx, dy, I;} CORNER_LIST[MAX_CORNERS];

class susan {
 INT x_size;             // ancho de imagen
 INT y_size;             // altura de imagen
 HBYTE *in;              //  imagen
 CORNER_LIST  corner_list;  //lista de esquinas
 HBYTE *bp;  //LUT (Look-up Table) de luminosidad

// llena bp (LUT) con valores de luminosidad según la fórmula
 int brightness(INT b) {        // b es umbral para la diferencia entre núcleo y pixel
        INT form = 6, k;   //  k es valor de gris
        float temp;
         for(k=-256;k<257;k++)
         {
                 temp=((float)k)/((float)b);
                 temp=temp*temp;
                 if (form==6)
                 temp=temp*temp*temp;
                 temp=(float)(100.0*exp(-temp));
                 *(bp+k)= (HBYTE)temp;
         }
         return 0;
 }
// Constructor
public:
 susan(HBYTE *image, INT n_x, INT n_y){   //imagen con sus tamaños
         INT i;
         x_size=n_x;
         y_size=n_y;
         in=(HBYTE *)malloc(x_size*y_size);  //espacio para imagen in
         if(in==0) exit(1);
         HBYTE *inp, *inp1;   //imágenes adicionales

         if(image==0) exit(1);
         inp=image;   //inp contiene imagen inicial
         inp1=in;  //inp1 contiene imagen vacío in
```

```
        for(i=0; i < y_size*x_size; i++)  {
                //reasignación de imágenes, inp1 tiene imagen inicial
                *inp1++=*inp;  inp++;
        }
        bp=new HBYTE[513];  //espacio para bp (LUT)
        bp=bp+256;  //apuntador a posición central de bp
 };  /* fin de  constructor*/
//
// dibujar esquinas
//  num es cantidad, row y col posiciones de esquinas, im_out imagen salida
 int cornerdraw(INT num, INT *row, INT *col, HBYTE *im_out){
        HBYTE *p;
        INT   i;
        HBYTE *inp, *inp1;

        if(in==0) return 1;
        inp=in;  //asignar imagen inicial a inp
        inp1=im_out;  //asignar imagen final a inp1
        for(i=0; i < y_size*x_size; i++)  {
       //reasignación de imágenes, inp1 tiene imagen inicial
                *inp1++=*inp;  inp++;
        }

        i = 0;
 while(i < num)    //dibujar esquinas sobre la imagen im_out
 {
     p = im_out + (row[i]-1)*x_size + col[i] - 1;
    *p++=255; *p++=255; *p=255; p+=x_size-2;
    *p++=255; *p++=0;   *p=255; p+=x_size-2;
    *p++=255; *p++=255; *p=255;
    i++;
  }
        return 0;
 };      // fin de cornerdraw
//
// lista de esquinas
// bt es umbral para la diferencia entre núcleo y pixel, num es cantidad de esquinas,
// row y col sus posiciones
 int cornersusanq(INT bt, INT &num, INT *row, INT *col)  {
 HBYTE                  *p, *cp;  /* p es apuntador para pixeles de imagen
                                     cp es apuntador para valores de LUT */
 INT                   *r;  // arreglo de cantidades de puntos seleccionados
 INT             i, j, n, x, y,max_no; //x, y posiciones de pixel
                                  //n es cantidad de puntos seleccionados
 brightness(bt);  //calcular LUT
  r  =  (INT *) calloc(x_size * y_size , sizeof(INT));  //asignación de espacio para r
 max_no = 1850;  //número máximo de esquinas
```

```
 // aplicar la máscara de SUSAN sobre la imagen
 for (i=7;i<y_size-7;i++)
   for (j=7;j<x_size-7;j++) {
      n=100;
      p=in + (i-3)*x_size + j - 1;  //apuntador a primer pixel bajo la máscara
      cp=bp + in[i*x_size+j];  //apuntador a un valor de LUT
      //desplazamiento de la máscara y cálculo de puntos seleccionados
      n+=*(cp-*p++);  n+=*(cp-*p++);  n+=*(cp-*p);
      p+=x_size-3;

      n+=*(cp-*p++);  n+=*(cp-*p++); n+=*(cp-*p++);
      n+=*(cp-*p++);  n+=*(cp-*p);
      p+=x_size-5;

      n+=*(cp-*p++);  n+=*(cp-*p++);  n+=*(cp-*p++);
      n+=*(cp-*p++);  n+=*(cp-*p++);  n+=*(cp-*p++);
      n+=*(cp-*p);
      p+=x_size-6;

      n+=*(cp-*p++);  n+=*(cp-*p++);  n+=*(cp-*p);
     if (n<max_no){ p+=2;  n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p);
     if (n<max_no){ p+=x_size-6;  n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p);
     if (n<max_no){ p+=x_size-5;  n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p);
     if (n<max_no){ p+=x_size-3; n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p++);
     if (n<max_no){ n+=*(cp-*p);
     if (n<max_no)  r[i*x_size+j] = max_no-n; //arreglo final r
}}}}}}}}}}}}}}}}}}

 /* localizar los máximos locales en el arreglo r */
 n=0;
 for (i=7;i<y_size-7;i++)
   for (j=7;j<x_size-7;j++) {
      x = r[i*x_size+j];  //una cantidad de esquinas de r
      if (x>0)  {
```

```
        /* 5x5 mask */   //desplazar la máscara 5x5
#ifdef FIVE_SUPP
        if (
           (x>r[(i-1)*x_size+j+2]) && (x>r[(i  )*x_size+j+1]) &&
           (x>r[(i  )*x_size+j+2]) && (x>r[(i+1)*x_size+j-1]) &&
           (x>r[(i+1)*x_size+j  ]) && (x>r[(i+1)*x_size+j+1]) &&
           (x>r[(i+1)*x_size+j+2]) && (x>r[(i+2)*x_size+j-2]) &&
           (x>r[(i+2)*x_size+j-1]) &&(x>r[(i+2)*x_size+j  ]) &&
           (x>r[(i+2)*x_size+j+1]) && (x>r[(i+2)*x_size+j+2]) &&
           (x>=r[(i-2)*x_size+j-2]) && (x>=r[(i-2)*x_size+j-1]) &&
           (x>=r[(i-2)*x_size+j  ]) &&(x>=r[(i-2)*x_size+j+1]) &&
           (x>=r[(i-2)*x_size+j+2]) && (x>=r[(i-1)*x_size+j-2]) &&
           (x>=r[(i-1)*x_size+j-1]) && (x>=r[(i-1)*x_size+j  ]) &&
           (x>=r[(i-1)*x_size+j+1]) &&(x>=r[(i  )*x_size+j-2]) &&
           (x>=r[(i  )*x_size+j-1]) && (x>=r[(i+1)*x_size+j-2]) )
#endif
#ifdef SEVEN_SUPP   //desplazar la máscara 7x7
        if (
            (x>r[(i-3)*x_size+j-3]) && (x>r[(i-3)*x_size+j-2]) &&
            (x>r[(i-3)*x_size+j-1]) && (x>r[(i-3)*x_size+j  ]) &&
            (x>r[(i-3)*x_size+j+1]) && (x>r[(i-3)*x_size+j+2]) &&
            (x>r[(i-3)*x_size+j+3]) && (x>r[(i-2)*x_size+j-3]) &&
            (x>r[(i-2)*x_size+j-2]) && (x>r[(i-2)*x_size+j-1]) &&
            (x>r[(i-2)*x_size+j  ]) && (x>r[(i-2)*x_size+j+1]) &&
            (x>r[(i-2)*x_size+j+2]) && (x>r[(i-2)*x_size+j+3]) &&
            (x>r[(i-1)*x_size+j-3]) && (x>r[(i-1)*x_size+j-2]) &&
            (x>r[(i-1)*x_size+j-1]) && (x>r[(i-1)*x_size+j  ]) &&
            (x>r[(i-1)*x_size+j+1]) && (x>r[(i-1)*x_size+j+2]) &&
            (x>r[(i-1)*x_size+j+3]) && (x>r[(i)*x_size+j-3]) &&
            (x>r[(i)*x_size+j-2]) && (x>r[(i)*x_size+j-1]) &&
            (x>=r[(i)*x_size+j+1]) && (x>=r[(i)*x_size+j+2]) &&
            (x>=r[(i)*x_size+j+3]) && (x>=r[(i+1)*x_size+j-3]) &&
            (x>=r[(i+1)*x_size+j-2]) && (x>=r[(i+1)*x_size+j-1]) &&
            (x>=r[(i+1)*x_size+j  ]) && (x>=r[(i+1)*x_size+j+1]) &&
             (x>=r[(i+1)*x_size+j+2]) && (x>=r[(i+1)*x_size+j+3]) &&
             (x>=r[(i+2)*x_size+j-3]) && (x>=r[(i+2)*x_size+j-2]) &&
            (x>=r[(i+2)*x_size+j-1]) && (x>=r[(i+2)*x_size+j  ]) &&
            (x>=r[(i+2)*x_size+j+1]) && (x>=r[(i+2)*x_size+j+2]) &&
            (x>=r[(i+2)*x_size+j+3]) && (x>=r[(i+3)*x_size+j-3]) &&
            (x>=r[(i+3)*x_size+j-2]) && (x>=r[(i+3)*x_size+j-1]) &&
            (x>=r[(i+3)*x_size+j  ]) && (x>=r[(i+3)*x_size+j+1]) &&
            (x>=r[(i+3)*x_size+j+2]) && (x>=r[(i+3)*x_size+j+3]) )
#endif
{  //llenar la estructura corner_list
corner_list[n].info=0;
corner_list[n].x=j;    //posiciones en la imagen
corner_list[n].y=i;
```

```
//x es mediana de esquina para calcular la intensidad
x = in[(i-2)*x_size+j-2] + in[(i-2)*x_size+j-1] + in[(i-2)*x_size+j] + in[(i-2)*x_size+j+1]
+ in[(i-2)*x_size+j+2] +
   in[(i-1)*x_size+j-2] + in[(i-1)*x_size+j-1] + in[(i-1)*x_size+j] + in[(i-1)*x_size+j+1] +
in[(i-1)*x_size+j+2] +
   in[(i   )*x_size+j-2] + in[(i   )*x_size+j-1] + in[(i   )*x_size+j] + in[(i   )*x_size+j+1] +
in[(i  )*x_size+j+2] +
   in[(i+1)*x_size+j-2] + in[(i+1)*x_size+j-1] + in[(i+1)*x_size+j] + in[(i+1)*x_size+j+1]
+ in[(i+1)*x_size+j+2] +
   in[(i+2)*x_size+j-2] + in[(i+2)*x_size+j-1] + in[(i+2)*x_size+j] + in[(i+2)*x_size+j+1]
+ in[(i+2)*x_size+j+2];

corner_list[n].I=x/25;   //intensidad
// x, y son valores para calcular el gradiente de intensidad
x    =    in[(i-2)*x_size+j+2]    +    in[(i-1)*x_size+j+2]    +    in[(i)*x_size+j+2]    +
in[(i+1)*x_size+j+2] + in[(i+2)*x_size+j+2] -
  (in[(i-2)*x_size+j-2] + in[(i-1)*x_size+j-2] + in[(i)*x_size+j-2] + in[(i+1)*x_size+j-2] +
in[(i+2)*x_size+j-2]);
x  +=  x  +  in[(i-2)*x_size+j+1]  +  in[(i-1)*x_size+j+1]  +  in[(i)*x_size+j+1]  +
in[(i+1)*x_size+j+1] + in[(i+2)*x_size+j+1] -
      (in[(i-2)*x_size+j-1] + in[(i-1)*x_size+j-1] + in[(i)*x_size+j-1] + in[(i+1)*x_size+j-
1] + in[(i+2)*x_size+j-1]);

y    =    in[(i+2)*x_size+j-2]    +    in[(i+2)*x_size+j-1]    +    in[(i+2)*x_size+j]    +
in[(i+2)*x_size+j+1] + in[(i+2)*x_size+j+2] -
  (in[(i-2)*x_size+j-2] + in[(i-2)*x_size+j-1] + in[(i-2)*x_size+j] + in[(i-2)*x_size+j+1] +
in[(i-2)*x_size+j+2]);
y  +=  y  +  in[(i+1)*x_size+j-2]  +  in[(i+1)*x_size+j-1]  +  in[(i+1)*x_size+j]  +
in[(i+1)*x_size+j+1] + in[(i+1)*x_size+j+2] -
      (in[(i-1)*x_size+j-2]    +    in[(i-1)*x_size+j-1]    +    in[(i-1)*x_size+j]    +    in[(i-
1)*x_size+j+1] + in[(i-1)*x_size+j+2]);
corner_list[n].dx=x/15;  // gradiente de intensidad
corner_list[n].dy=y/15;
n++;
if(n==MAX_CORNERS){
        cout << "Too many corners" << endl;
        return 1;
      }}}}
corner_list[n].info=7;   //tipo de máscara usada
n=0;
do {
        row[n]=corner_list[n].y;   //posición de la esquina
        col[n]=corner_list[n].x;
        n++;
} while(!(corner_list[n].info==7));
        num=n;          //cantidad de esquinas
return 0;
```

```
 }                  // fin de cornersusanq
} //fin  clase
```

Pseudocódigo del método basado en algoritmo SUSAN.

```
/*lee la imagen de un archivo */
read_image (Ip071, 'E:\\Archivos de
programa\\MVTec\\Halcon\\images\\bvlab_2p\\2p_071.tif')
/*obtiene el apuntador con el tipo de la imagen y su tamaño */
get_image_pointer1 (Ip071, Pointer, Type, Width, Height)
read_image (Ip072, 'E:\\Archivos de
programa\\MVTec\\Halcon\\images\\bvlab_2p\\2p_072.tif')
/*abre la ventana para la imagen */
open_window(0,0,Width,Height,0,"visible","",&WindowID);
  /*encuentra las esquinas de la imagen */
user_cornerq (Ip071, 50, Rows, Cols)
/*dibuja las esquinas encontradas */
user_cornerdraw (Ip071, ImageOut, Rows, Cols)
/*despliega la imagen */
disp_image(ImageOut,WindowID);
/*espera el clic en la ventana */
get_mbutton(WindowID,&Row,&Column,&Button);
/*escribe la imagen a un archivo */
write_image (ImageOut, 'tiff', 0, 'E:\\Tania\\halcon\\MatchImage\\corner071')
user_cornerq (Ip072, 50, Rows1, Cols1)
user_cornerdraw (Ip072, ImageOut1, Rows1, Cols1)
disp_image(ImageOut1,WindowID);
get_mbutton(WindowID,&Row,&Column,&Button);
write_image (ImageOut1, 'tiff', 0, 'E:\\Tania\\halcon\\MatchImage\\corner072')
/*pone al vacío la región de unión */
gen_empty_region (RegionUnion)
for i := 0 to |Rows|-1 by 1
       /*genera el círculo en la posición asignada */
      gen_circle (Circle, Rows[i], Cols[i], 2)
       /*une la región  y el círculo*/
       union2 (RegionUnion, Circle, RegionUnion)
endfor
/*pinta la región final */
paint_region (RegionUnion, ImageOut1, ImageResult, 5, 'fill')
write_image (ImageResult, 'tiff', 0, 'E:\\tania\\halcon\\MatchImage\\cornerUnion')
disp_image(ImageResult,WindowID);
get_mbutton(WindowID,&Row,&Column,&Button);
/*cierra la ventana */
close_window(WindowID);
```