Apéndice C

Programa

```
//---------------------------------------------------------------------------
---
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include <math.h>
#include "Unit2.h"
#include "Unit3.h"
#include "OpenGL.h"
//---------------------------------------------------------------------------
---
#pragma package(smart_init)
#pragma link "VCap"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
---
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  //mascara sobel 3x3
  sx[1][1]=-1.0;
  sx[1][2]=0.0;
  sx[1][3]=1.0;
  sx[2][1]=-2.0;
  sx[2][2]=0.0;
  sx[2][3]=2.0;
  sx[3][1]=-1.0;
  sx[3][2]=0.0;
  sx[3][3]=1.0;

  sy[1][1]=1.0;
  sy[1][2]=2.0;
  sy[1][3]=1.0;
  sy[2][1]=0.0;
  sy[2][2]=0.0;
  sy[2][3]=0.0;
  sy[3][1]=-1.0;
  sy[3][2]=-2.0;
  sy[3][3]=-1.0;
  //inicialización tamano imagenes
  max_img_x=354;
  max_img_y=290;
  //inicializacion pinta lineas detectadas de hough
  PintaBorde1->Checked=false;  //dibuja vectores de transf
hough
  FiltradoON->Checked=false;   //realiza o no filtrado
pasabajas
  BordeCanny1->Checked=true;   //realiza detector de bordes
canny o sobel
  Caracterist->Checked=true;   //aparece resultado de
coordenadas y matching en pantalla
  Sigmoidal->Checked=true;     //habilita filtrado sigmoidal

  exponencial->Checked=false;
  polinomio->Checked=false;
  noajuste->Checked=true;
  //inicializacion thresholds
  thres=40;          //threshold binarizacion
  thres_h=5;         //threshold tamano de no muestras para
hough
  thres_z=8;         //threshold zonas de hough
  thres_c=5000;
  thres_s=128;
  thres_cam=7;
  thres_lin=8;
  thres_mul=26;
  //inicializacion relleno canvas
  Image1->Canvas->Brush->Color=clBlue;
  Image1->Canvas->FillRect(Rect(0,0,Image1-
>Width,Image1->Height));
  Image2->Canvas->Brush->Color=clBlue;
  Image2->Canvas->FillRect(Rect(0,0,Image2-
>Width,Image2->Height));
  Lienzo1->Canvas->Brush->Color=clBlack;
  Lienzo1->Canvas->FillRect(Rect(0,0,Lienzo1-
>Width,Lienzo1->Height));
  Lienzo2->Canvas->Brush->Color=clBlack;
  Lienzo2->Canvas->FillRect(Rect(0,0,Lienzo2-
>Width,Lienzo2->Height));
  Lienzo3->Canvas->Brush->Color=clWhite;
  Lienzo3->Canvas->FillRect(Rect(0,0,Lienzo3-
>Width,Lienzo3->Height));
  Lienzo4->Canvas->Brush->Color=clWhite;
  Lienzo4->Canvas->FillRect(Rect(0,0,Lienzo4-
>Width,Lienzo4->Height));
  //inicialiacion colores corregidos
  r_cal_1=0.68;
  g_cal_1=0.80;
  b_cal_1=1.0;
  r_cal_2=0.77;
  g_cal_2=0.82;
  b_cal_2=1.0;

  a="ICM532A";
  if (!Video->Init()) {
    ShowMessage("Can't init capture!");
    Application->Terminate();
  };
  config = new TGraphConfig();
  Video->SaveGraph(config);
  config->Clear();

  config->VCapSource=a;
  config->WantPreview=true;
  config->WantCapture=true;
  config->WantBitmaps=true;
  //config->CaptureFileName="aa.avi";
  Video->RestoreGraph(config);
}
//---------------------------------------------------------------------------
---
void __fastcall
TForm1::VideoBitmapGrabbed(TCapturedBitmap
*CapturedImage)
{
  CapturedImage->SaveToFile(nombre_foto);
  if(nombre_foto=="Img1.bmp")Image1->Picture-
>Assign(CapturedImage);
  if(nombre_foto=="Img2.bmp")Image2->Picture-
>Assign(CapturedImage);
}
//---------------------------------------------------------------------------
---
void TForm1::accion1()
{
  a="ICM532A";

  Video->SaveGraph(config);
    config->VCapSource=a;
  Video->RestoreGraph(config);
```

```
  Video->StartPreview();

  nombre_foto="Img1.bmp";
  Video->CaptureFrame();
}
//---------------------------------------------------------------------------
---
void TForm1::accion2()
{
  a="ICM532B";

  Video->SaveGraph(config);
    config->VCapSource=a;
  Video->RestoreGraph(config);

  Video->StartPreview();

  nombre_foto="Img2.bmp";
  Video->CaptureFrame();
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::Threshold1Click(TObject *Sender)
{
  Form2=new TForm2 (Application);
  Form2->ShowModal();
  delete Form2;
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::DetenerClick(TObject
*Sender)//detiene captura
{
  Timer1->Interval=0;
  Timer2->Interval=0;

  Detener->Enabled=false;
  Accion1->Enabled=true;
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::Accion1Click(TObject
*Sender)//inicia captura
{
  Timer1->Interval=2500;
  Sleep(1250);
  Timer2->Interval=2500;

  Detener->Enabled=true;
  Accion1->Enabled=false;
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::StartButtonClick(TObject *Sender)
{
  Timer1->Interval=2500;
  Sleep(1250);
  Timer2->Interval=2500;

  Detener->Enabled=true;
  Accion1->Enabled=false;
}
//---------------------------------------------------------------------------
-
void __fastcall TForm1::Timer2Timer(TObject *Sender)
{
  accion2();           //toma la imagen 2
  exponential_adj(2);   //contraste
  low_pass(2);         //filtro difuminado
  gray_borde_bin(2);
  Hough(2);            //si es igual a 1 mando a llamar algoritmo
2Dto3D (vectorizo)
  color(2);
  Matching();          //empalme de las imagenes
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  accion1();           //toma imagen 1
  exponential_adj(1);   //contraste
  low_pass(1);         //filtro difuminado
  gray_borde_bin(1);
  Hough(1);            //vectoriza
  color(1);
}
//---------------------------------------------------------------------------
---
void TForm1::iniimg(TPicture* pBitmap,TPicture*
pBitmap2,int numero)//1 o 2 color
{
  if(numero==1)pBitmap->Bitmap-
>LoadFromFile("Img1.bmp");
  if(numero==2)pBitmap->Bitmap-
>LoadFromFile("Img2.bmp");

  pBitmap2->Bitmap->LoadFromFile("aaa.bmp");

  pBitmap->Bitmap->PixelFormat=pf24bit,pf24bit;
  pBitmap2->Bitmap->PixelFormat=pf24bit,pf24bit;
}
//---------------------------------------------------------------------------
---
void TForm1::iniimg2(TPicture* pBitmap,TPicture*
pBitmap2)//2 imagenes sobel
{
  pBitmap->Bitmap->LoadFromFile("Img1s.bmp");
  pBitmap2->Bitmap->LoadFromFile("Img2s.bmp");

  pBitmap->Bitmap->PixelFormat=pf24bit,pf24bit;
  pBitmap2->Bitmap->PixelFormat=pf24bit,pf24bit;
}
//---------------------------------------------------------------------------
---
void TForm1::DrawLinea(int x,int y,int x1,int y1,float b,float
g,float r,TPicture* pBitmap,int canv)
{
  int dx=x1-x;
  int dy=y1-y;
  int steps;
  float xinc;
  float yinc;
  float xp=0,yp=0;

  int xmax=pBitmap->Bitmap->Width;
  int  ymax=pBitmap->Bitmap->Height;
  int  cx=xmax/2;
  int  cy=ymax/2;

  if((abs(dx))>=(abs(dy))) steps=abs(dx);
  if((abs(dy))>(abs(dx))) steps=abs(dy);

  if(canv==0)Lienzo->Canvas->Pixels[cx+x][ymax-cy-
y]=(b*65536)+(g*256)+(r);
  //if(canv==1)Lienzo1->Canvas->Pixels[cx+x][ymax-cy-
y]=(b*65536)+(g*256)+(r);
  //if(canv==2)Lienzo2->Canvas->Pixels[cx+x][ymax-cy-
y]=(b*65536)+(g*256)+(r);
  if(canv==3)Lienzo3->Canvas->Pixels[cx+x][ymax-cy-
y]=(b*65536)+(g*256)+(r);
```

```
  if(canv==4)Lienzo4->Canvas->Pixels[cx+x][ymax-cy-
y]=(b*65536)+(g*256)+(r);

  if(steps>0)
  {
   xinc=(dx/steps)+(((dx%steps)*10000)/steps*.0001);
   yinc=(dy/steps)+(((dy%steps)*10000)/steps*.0001);

   xp=x;
   yp=y;
   for(int k=1;k<=steps;k++)
    {
     xp+=xinc;
     yp+=yinc;

     if(canv==0)Lienzo->Canvas->Pixels[(cx+xp)][(ymax-cy-
yp)]=(b*65536)+(g*256)+(r);
     //if(canv==1)Lienzo1->Canvas->Pixels[(cx+xp)][(ymax-
cy-yp)]=(b*65536)+(g*256)+(r);
     //if(canv==2)Lienzo2->Canvas->Pixels[(cx+xp)][(ymax-
cy-yp)]=(b*65536)+(g*256)+(r);
     if(canv==3)Lienzo3->Canvas->Pixels[(cx+xp)][(ymax-cy-
yp)]=(b*65536)+(g*256)+(r);
     if(canv==4)Lienzo4->Canvas->Pixels[(cx+xp)][(ymax-cy-
yp)]=(b*65536)+(g*256)+(r);
    }
  }
}
//---------------------------------------------------------------------------
---
void TForm1::polar(int r,int t,TPicture* pBitmap,int
img)//diuja linea polar
{
  float angulo,m;
  int x,y,x2,y2;

  angulo=t*M_PI/180.0;                  //angulo en radianes
  x2=floor(cos(angulo)*r);              //coordenada x de polar
  y2=floor(sin(angulo)*r);              //coordenada y de polar

  angulo=angulo+(M_PI/2.0);
  m=tan(angulo);                        //obtener punto y pendiente
  angulo=fmod(angulo,(2*M_PI));         //modulo flotante
                                //con punto y pendiente y una
interseccion con
                                //los ejes obtener la coordenada
completa
  if((angulo>=0)&&(angulo<(M_PI/4.0)))
  {
   x=-200;
   y=y2-((x2-x)*m);
  }
  if((angulo>=(M_PI/4.0))&&(angulo<(M_PI/2.0)))
  {
   y=200;
   x=(x2-((y2-y)/m));
  }

if((angulo>=(M_PI/2.0))&&(angulo<((M_PI/2.0)+(M_PI/4.0))
))
  {
   y=-200;
   x=(x2-((y2-y)/m));
  }
  if((angulo>=((M_PI/2.0)+(M_PI/4.0)))&&(angulo<(M_PI)))
  {
   x=-200;
   y=y2-((x2-x)*m);
  }
  if((angulo>=(M_PI))&&(angulo<(M_PI+(M_PI/4.0))))
  {
   x=200;
   y=y2-((x2-x)*m);
  }

if((angulo>=(M_PI+(M_PI/4.0)))&&(angulo<(M_PI+(M_PI/2
.0))))
  {
   y=-200;
   x=(x2-((y2-y)/m));
  }

if((angulo>=(M_PI+(M_PI/2.0)))&&(angulo<(M_PI+(M_PI/2
.0)+(M_PI/4.0))))
  {
   y=200;
   x=(x2-((y2-y)/m));
  }

if((angulo>=(M_PI+(M_PI/2.0)+(M_PI/4.0)))&&(angulo<(2.0
*M_PI)))
  {
   x=200;
   y=y2-((x2-x)*m);
  }
  if(img==1)DrawLinea(x,y,x2,y2,0,0,255,pBitmap,3);
  if(img==2)DrawLinea(x,y,x2,y2,0,0,255,pBitmap,4);
}
//---------------------------------------------------------------------------
--
void TForm1::get_pol(int x,int y,float angulo,TPicture*
pBitmap)
{
  float angulo2,x3,y3;                  //para hacer calculos en
flotante
  int t,r;
  int x2,y2;

  angulo2=angulo+(M_PI/2.0);            //obtiene angulo
perpendicular
  if(angulo2>(M_PI+0.1))angulo2=angulo2-(M_PI);   //para
trabajar solo con angulos positivos de 0 a 180

  x2=floor((y-(x*tan(angulo)))/(tan(angulo2)-tan(angulo))+.5);
//obtiene coord x de coord. polar
  x3=(y-(x*tan(angulo)))/(tan(angulo2)-tan(angulo));
//obtiene x flotante

  y2=floor((x3*tan(angulo2))+.5);       //obtiene y
  y3=x3*tan(angulo2);                   //obtiene y flotante

  if(x2==0)y2=y-((x)*tan(angulo));      //si x=0 se obtiene y
por otra formula

  if((y2)>800)y2=800;                   //evita que y se diapare
  if((y2)<-800)y2=-800;

  //if((x2==0)&&(y2==0))y2=1;
  r=sqrt((x3*x3)+(y3*y3));              //obtiene radio

  if(y2<0.0)angulo2+=M_PI;     //evita radios negativos
  if((x>=0)&&(angulo>=(M_PI/2.0)-
0.2)&&(angulo<=(M_PI/2.0)+0.2))angulo2=0.0;  //si hay
lineas verticales en x positivas las dibuja en polar como 0

  t=floor((angulo2*180.0/(M_PI))+0.5);      //angulo en rad a
grad para arreglo
```

```
  //Errors->Lines->Add("X:"+IntToStr(x)+"|
Y:"+IntToStr(y)+"| Angulo:"+FloatToStr(angulo2)+"|
R:"+IntToStr(r));
  if((r<600)&&(t>=0)&&(t<361))                    //evita
desbordamiento
    arreglo[t][r]++;                              //manda a arreglo
}
//---------------------------------------------------------------------------
---
void TForm1::get_rt(int x,int y,float angulo,TPicture*
pBitmap)
{
  float angulo2,x3,y3;                        //para hacer calculos en
flotante
  int t,r;
  int x2,y2;

  angulo2=angulo+(M_PI/2.0);                  //obtiene angulo
perpendicular
  if(angulo2>(M_PI+0.1))angulo2=angulo2-(M_PI);   //para
trabajar solo con angulos positivos de 0 a 180

  x2=floor((y-(x*tan(angulo)))/(tan(angulo2)-tan(angulo))+.5);
//obtiene coord x de coord. polar
  x3=(y-(x*tan(angulo)))/(tan(angulo2)-tan(angulo));
//obtiene x flotante

  y2=floor((x3*tan(angulo2))+.5);             //obtiene y
  y3=x3*tan(angulo2);                         //obtiene y flotante

  if(x2==0)y2=y-((x)*tan(angulo));            //si x=0 se obtiene y
por otra formula

  if((y2)>800)y2=800;                         //evita que y se diapare
  if((y2)<-800)y2=-800;

  //if((x2==0)&&(y2==0))y2=1;
  r=sqrt((x3*x3)+(y3*y3));                    //obtiene radio

  if(y2<0.0)angulo2+=M_PI;   //evita radios negativos
  if((x>=0)&&(angulo>=(M_PI/2.0)-
0.2)&&(angulo<=(M_PI/2.0)+0.2))angulo2=0.0; //si hay lineas
verticales en x positivas las dibuja en polar como 0

  t=floor((angulo2*180.0/(M_PI))+0.5);        //angulo en rad a
grad para arreglo

  //Errors->Lines->Add("X:"+IntToStr(x)+"|
Y:"+IntToStr(y)+"| Angulo:"+FloatToStr(angulo2)+"|
R:"+IntToStr(r));
  rad=r;
  tet=t;
}
//---------------------------------------------------------------------------
---
int TForm1::diferencia(int x0i,int x1i,int x2i,int x0d,int x1d,int
x2d,int xi,int xd,int y,int menor)
{ //hace la diferencia entre 2 características
  int result;
  float inci,incd;      //gurada angulo de características

  //hacer un recorrido en las 30 casillas a ver cual es true
  //indicando que ese es el angulo de ese pixel
  for(int ciclo=1;ciclo<=thres_z*2;ciclo++)
  {
    if(s_arr[0][xi][y][ciclo-1]==true)
      inci=((2.0*M_PI)/(thres_z*4))*ciclo;
    if(s_arr[1][xd][y][ciclo-1]==true)
      incd=((2.0*M_PI)/(thres_z*4))*ciclo;
  }

    //result=abs(x1i-x1d)+abs(x0i-x0d)+abs(x2i-x2d);
//resultado tomando en cuenta el pixel central
    //result=abs(x0i-x0d)+abs(x2i-x2d);  //resultado sin tomar en
cuenta el pixel central y sin inclinacion
  result=abs(x0i-x0d)+abs(x2i-x2d)+abs(inci-incd);//resultado
sin tomar en cuenta el pixel central CON inclinacion

  return result;
}
//---------------------------------------------------------------------------
---
void TForm1::dosDtotresD(TPicture* pBitmap)//coordenadas
2D a coordenadas 3D
{
  float X,Y,Z;
  int xr,xl,y;
  int cx,cy;
  int stereo_disp;
  double sd;
  float D=thres_cam;
  float f=10.0;

  //pBitmap->Bitmap->LoadFromFile("Img1.bmp");

  Lienzo->Canvas->Brush->Color=clWhite;
  Lienzo->Canvas->FillRect(Rect(0,0,Lienzo->Width,Lienzo-
>Height));
  //dibuja ejes no es necesario correr ya lo posiciona
  DrawLinea(100,0,-100,0,255,0,0,pBitmap,0);
  DrawLinea(0,100,0,-100,255,0,0,pBitmap,0);

  cy=pBitmap->Bitmap->Height/2;
  cx=pBitmap->Bitmap->Width/2;

  Errors->Lines->Add("------------------- 3D ----------------");
  for(int ciclo1=0;ciclo1<ptr_pares;ciclo1++)//recorre arreglo
pares
  {
    xl=pares[ciclo1][0];
    xr=pares[ciclo1][2];
    y=pares[ciclo1][1];

    stereo_disp=xl-xr;
    if(stereo_disp>0)
    {
      sd=stereo_disp;
      //convertir a coordenadas cartesianas
      xl=xl-cx;
      xr=xr-cx;
      y=cy-y;
      //formula X
      X=floor(((D*(xr+xl))/(2.0*(xl-xr)))+.5);
      //formula Y
      Y=floor(((D*y)/(xl-xr))+0.5);
      //formula Z
      //Z=floor(((7*650.0)/(xl-xr))+0.5);

      if(thres_cam==7)
      {
        if(polinomio->Checked==true)
        Z=470.787854-
(16.60765846*sd)+(0.268101545*pow(sd,2.0))+(-
0.002120258*pow(sd,3.0))+(7.79504E-06*pow(sd,4.0))+(-
1.04632E-08*pow(sd,5.0));
        if(exponencial->Checked==true)
        Z=floor(((D*f)/(xl-xr))+0.5);
        if(noajuste->Checked==true)
        Z=floor(((7*575.0)/(xl-xr))+0.5);
      }
```

```
      if(thres_cam==10)
      {
       if(polinomio->Checked==true)
       Z=933.7362657-
(30.63300217*sd)+(0.438465394*pow(sd,2.0))+(-
0.003084642*pow(sd,3.0))+(1.03127E-05*pow(sd,4.0))+(-
1.28933E-08*pow(sd,5.0));
       if(exponencial->Checked==true)
       Z=floor(((D*f)/(xl-xr))+0.5);
       if(noajuste->Checked==true)
       Z=floor(((10*700.0)/(xl-xr))+0.5);
      }
      if(thres_cam==14)
      {
       if(polinomio->Checked==true)
       Z=252.1291429-
(2.375071302*sd)+(0.003003327*pow(sd,2.0))+(7.62778E-
05*pow(sd,3.0))+(-4.13927E-07*pow(sd,4.0))+(6.17741E-
10*pow(sd,5.0));
       if(exponencial->Checked==true)
       Z=floor(((D*f)/(xl-xr))+0.5);
       if(noajuste->Checked==true)
       Z=floor(((14*620.0)/(xl-xr))+0.5);
      }

     //guarda coordenadas 3D
      Coord3D[ciclo1][0]=X;
      Coord3D[ciclo1][1]=Y;
      Coord3D[ciclo1][2]=Z;
      //no_coord=ptr_pares;

      Errors->Lines->Add("X="+FloatToStr(X)+"   | Y="+Y+"
| Z="+Z);
      //dibuja donde se encuentra la X real
      X=X*4.0;
      Y=Y*4.0;
      Z=Z*((double(thres_mul))/10.0);

      DrawLinea(X-4,Y,X+4,Y,Z,Z,Z,pBitmap,0);
      DrawLinea(X,Y-4,X,Y+4,Z,Z,Z,pBitmap,0); //color denota
profundidad
    }
  }
  Status->SimpleText=FloatToStr(Z)+"=Z  "+X+"=X
"+Y+"=Y   "+ptr_pares+" puntero pares";
  Status->SimpleText="Distancia entre
camaras="+FloatToStr(D)+"  ";
  Errors->Lines->SaveToFile("out.txt");
}
//---------------------------------------------------------------------------
---
void TForm1::color(int img)
{
  float a1,a2,a3,b1,b2,b3,f;
  float r=0.0,g=0.0,b=0.0;

  Graphics::TPicture *pBitmap = new Graphics::TPicture();
  Graphics::TPicture *pBitmap2 = new Graphics::TPicture();

  iniimg(pBitmap,pBitmap2,img);

  /*  for (int y = 1; y < pBitmap->Bitmap->Height-1; y++)
    {
     ptr0 = (Byte *)pBitmap->Bitmap->ScanLine[y-1];
     ptr1 = (Byte *)pBitmap->Bitmap->ScanLine[y];
     ptr2 = (Byte *)pBitmap->Bitmap->ScanLine[y+1];

     ptr = (Byte *)pBitmap2->Bitmap->ScanLine[y];
      //ptr3 =(Byte *)pBitmap2->Bitmap->ScanLine[y-1];
      for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3; x++)
      {

        //hace hough ( deteccion de bordes)

        a1=(ptr0[x-
3]*sx[1][1]+ptr0[x]*sx[1][2]+ptr0[x+3]*sx[1][3]+ptr1[x-
3]*sx[2][1]+ptr1[x]*sx[2][2]+ptr1[x+3]*sx[2][3]+ptr2[x-
3]*sx[3][1]+ptr2[x]*sx[3][2]+ptr2[x+3]*sx[3][3]);
        a2=(ptr0[x-
2]*sx[1][1]+ptr0[x+1]*sx[1][2]+ptr0[x+4]*sx[1][3]+ptr1[x-
2]*sx[2][1]+ptr1[x+1]*sx[2][2]+ptr1[x+4]*sx[2][3]+ptr2[x-
2]*sx[3][1]+ptr2[x+1]*sx[3][2]+ptr2[x+4]*sx[3][3]);
        a3=(ptr0[x-
1]*sx[1][1]+ptr0[x+2]*sx[1][2]+ptr0[x+5]*sx[1][3]+ptr1[x-
1]*sx[2][1]+ptr1[x+2]*sx[2][2]+ptr1[x+5]*sx[2][3]+ptr2[x-
1]*sx[3][1]+ptr2[x+2]*sx[3][2]+ptr2[x+5]*sx[3][3]);

        b1=(ptr0[x-
3]*sy[1][1]+ptr0[x]*sy[1][2]+ptr0[x+3]*sy[1][3]+ptr1[x-
3]*sy[2][1]+ptr1[x]*sy[2][2]+ptr1[x+3]*sy[2][3]+ptr2[x-
3]*sy[3][1]+ptr2[x]*sy[3][2]+ptr2[x+3]*sy[3][3]);
        b2=(ptr0[x-
2]*sy[1][1]+ptr0[x+1]*sy[1][2]+ptr0[x+4]*sy[1][3]+ptr1[x-
2]*sy[2][1]+ptr1[x+1]*sy[2][2]+ptr1[x+4]*sy[2][3]+ptr2[x-
2]*sy[3][1]+ptr2[x+1]*sy[3][2]+ptr2[x+4]*sy[3][3]);
        b3=(ptr0[x-
1]*sy[1][1]+ptr0[x+2]*sy[1][2]+ptr0[x+5]*sy[1][3]+ptr1[x-
1]*sy[2][1]+ptr1[x+2]*sy[2][2]+ptr1[x+5]*sy[2][3]+ptr2[x-
1]*sy[3][1]+ptr2[x+2]*sy[3][2]+ptr2[x+5]*sy[3][3]);

        ptr[x]=sqrt(a1*a1+b1*b1);
        ptr[x+1]=sqrt(a2*a2+b2*b2);
        ptr[x+2]=sqrt(a3*a3+b3*b3);


        //Status->SimpleText="OK";
        //realiza binarizacion con threshold
        f=0.299*ptr[x+2]+0.587*ptr[x+1]+0.114*ptr[x];
        if(f<thres)
        {

        }
        else
        {
         r+=ptr1[x+2];
         g+=ptr1[x+1];
         b+=ptr1[x];

        }

        x=x+2;
      }
     }*/

    //hace pasada a toda la imágen
     for (int y = 1; y < pBitmap->Bitmap->Height-1; y++)
     {
      Form1->ptr1 = (Byte *)pBitmap->Bitmap->ScanLine[y];

      //Form1->ptr2 = (Byte *)pBitmap2->Bitmap-
>ScanLine[y];

      for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3; x++)
      {

         r+=Form1->ptr1[x+2];
         g+=Form1->ptr1[x+1];
         b+=Form1->ptr1[x];

        // r2+=Form1->ptr2[x+2];
```

```
        // g2+=Form1->ptr2[x+1];
        // b2+=Form1->ptr2[x];

       x=x+2;
     }
    }
    //Status->SimpleText="OK";
    if(img==1)
    {
     R=r*r_cal_1;
     G=g*g_cal_1;
     B=b*b_cal_1;
    }
    if(img==2)
    {
     R2=r*r_cal_2;
     G2=g*g_cal_2;
     B2=b*b_cal_2;
    }

    Status->SimpleText="r="+FloatToStr(r)+" | g="+g+" |
b="+b+"R="+  R+" | G="+G+" | B="+B;
    LienzoColor->Canvas->Brush->Color=clWhite;
    R=(R+R2)/2.0;
    G=(G+G2)/2.0;
    B=(B+B2)/2.0;

    if((R>(G+(thres_c*10)))&&(B>(G+(thres_c*10))))
LienzoColor->Canvas->Brush->Color=clPurple;
    if((G>(R+(thres_c*10)))&&(B>(R+(thres_c*10))))
LienzoColor->Canvas->Brush->Color=clLime;
    if((R>(B+(thres_c*10)))&&(G>(B+(thres_c*10))))
LienzoColor->Canvas->Brush->Color=clYellow;


    if((R>(G+(thres_c*10)))&&(R>(B+(thres_c*10))))
LienzoColor->Canvas->Brush->Color=clRed;
    if((G>(R+(thres_c*10)))&&(G>(B+(thres_c*10))))
LienzoColor->Canvas->Brush->Color=clGreen;
    if((B>(R+(thres_c*10)))&&(B>(G+(thres_c*10))))
LienzoColor->Canvas->Brush->Color=clBlue;

  LienzoColor->Canvas->FillRect(Rect(0,0,LienzoColor-
>Width,LienzoColor->Height));

  delete pBitmap;
  delete pBitmap2;
}
//---------------------------------------------------------------------------
---
void TForm1::exponential_adj(int img) //ajuste exponencial
tipo sigmoidal
{
 if(Sigmoidal->Checked==true)
 {
  Graphics::TPicture *pBitmap = new Graphics::TPicture();
  Graphics::TPicture *pBitmap2 = new Graphics::TPicture();
  float f;
  int R,G,B;

   iniimg(pBitmap,pBitmap2,img);


   for (int y = 0; y < pBitmap->Bitmap->Height-1; y++)
    {

     ptr = (Byte *)pBitmap->Bitmap->ScanLine[y];

     for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3; x++)
     {

        //Errors->Lines->Add(ptr[x]);
        //componentes rgb
        R=ptr[x]*((1.0/(1.0+exp(-1*((ptr[x]-128.0)/thres_s))
))+0.5);
        G=ptr[x+1]*((1.0/(1.0+exp(-1*((ptr[x+1]-
128.0)/thres_s)) ))+0.5);
        B=ptr[x+2]*((1.0/(1.0+exp(-1*((ptr[x+2]-
128.0)/thres_s)) ))+0.5);

        if(R>255)R=255;
        if(G>255)G=255;
        if(B>255)B=255;

        ptr[x]=R;
        ptr[x+1]=G;
        ptr[x+2]=B;


        //ptr[x]=ptr[x]*(1.0/thres*.01);
        //ptr[x+1]=ptr[x+1]*(1.0/thres*.01);
        //ptr[x+2]=ptr[x+2]*(1.0/thres*.01);


       x=x+2;
     }
    }
    if(img==1)
    {
     Image1->Canvas->Draw(0,0,pBitmap->Bitmap);
     Image1->Repaint();
     pBitmap->SaveToFile("Img1.bmp");
    }
    if(img==2)
    {
     Image2->Canvas->Draw(0,0,pBitmap->Bitmap);
     Image2->Repaint();
     pBitmap->SaveToFile("Img2.bmp");
    }
  delete pBitmap;
  delete pBitmap2;
  }
}
//---------------------------------------------------------------------------
-
void TForm1::low_pass(int img) //filtrado smoothing
{
  float mask[5][5];

  if(FiltradoON->Checked==true)
  {
    if(Gaussiano->Checked==true) //realiza filtrado gaussiano
    {

      Graphics::TPicture *pBitmap = new Graphics::TPicture();
      Graphics::TPicture *pBitmap2 = new Graphics::TPicture();
      int arreglo[9],tmp;
      //iniimg(pBitmap,pBitmap2);
      iniimg(pBitmap,pBitmap2,img);


      for (int y = 1; y < pBitmap->Bitmap->Height-1; y++)
      {

        ptr0 = (Byte *)pBitmap->Bitmap->ScanLine[y-1];
        ptr = (Byte *)pBitmap->Bitmap->ScanLine[y];
        ptr2 = (Byte *)pBitmap->Bitmap->ScanLine[y+1];

        ptr1 = (Byte *)pBitmap2->Bitmap->ScanLine[y];
```

```
      for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3;
x++)
      {
       for(int color=0;color<3;color++)//ciclo para cada color
       {

         arreglo[0]=ptr0[x-3+color];
         arreglo[1]=ptr0[ x+color];
         arreglo[2]=ptr0[x+3+color];
         arreglo[3]=ptr[x-3+color];
         arreglo[4]=ptr[x+color];
         arreglo[5]=ptr[x+3+color];
         arreglo[6]=ptr2[x-3+color];
         arreglo[7]=ptr2[ x+color];
         arreglo[8]=ptr2[x+3+color];

         for(int cic1=0;cic1<9;cic1++)
         {
          for(int cic2=0;cic2<9;cic2++)
          {
            if(arreglo[cic1]>arreglo[cic2])//realiza ordenamiento
p/mediana
            {
             tmp=arreglo[cic1];
             arreglo[cic1]=arreglo[cic2];
             arreglo[cic2]=tmp;
            }
          }
         }
        ptr1[x+color]=arreglo[4];
       }//fin color
      x=x+2;
    }//x
   }//y

    if(img==1)
    {
     Image1->Canvas->Draw(0,0,pBitmap->Bitmap);
     Image1->Repaint();
     pBitmap2->SaveToFile("Img1.bmp");
    }
    if(img==2)
    {
     Image2->Canvas->Draw(0,0,pBitmap->Bitmap);
     Image2->Repaint();
     pBitmap2->SaveToFile("Img2.bmp");
    }


    delete pBitmap;
    delete pBitmap2;
   }



   else //si no es activo filtrado gaussiano hace pasabajas
   {
    mask[1][1]=1.0/16.0;
    mask[1][2]=1.0/8.0;
    mask[1][3]=1.0/16.0;
    mask[2][1]=1.0/8.0;
    mask[2][2]=1.0/4.0;
    mask[2][3]=1.0/8.0;
    mask[3][1]=1.0/16.0;
    mask[3][2]=1.0/8.0;
    mask[3][3]=1.0/16.0;

    Graphics::TPicture *pBitmap = new Graphics::TPicture();
    Graphics::TPicture *pBitmap2 = new Graphics::TPicture();
    iniimg(pBitmap,pBitmap2,img);

    for (int y = 1; y < pBitmap->Bitmap->Height-1; y++)
    {

     ptr0 = (Byte *)pBitmap->Bitmap->ScanLine[y-1];
     ptr = (Byte *)pBitmap->Bitmap->ScanLine[y];
     ptr2 = (Byte *)pBitmap->Bitmap->ScanLine[y+1];


      for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3;
x++)
      {
       ptr[x]=floor(ptr0[x-
3]*mask[1][1]+ptr0[x]*mask[1][2]+ptr0[x+3]*mask[1][3]+ptr
[x-
3]*mask[2][1]+ptr[x]*mask[2][2]+ptr[x+3]*mask[2][3]+ptr2[
x-
3]*mask[3][1]+ptr2[x]*mask[3][2]+ptr2[x+3]*mask[3][3]+.5);
        ptr[x+1]=floor(ptr0[x-
2]*mask[1][1]+ptr0[x+1]*mask[1][2]+ptr0[x+4]*mask[1][3]+
ptr[x-
2]*mask[2][1]+ptr[x+1]*mask[2][2]+ptr[x+4]*mask[2][3]+ptr
2[x-
2]*mask[3][1]+ptr2[x+1]*mask[3][2]+ptr2[x+4]*mask[3][3]+.
5);
        ptr[x+2]=floor(ptr0[x-
1]*mask[1][1]+ptr0[x+2]*mask[1][2]+ptr0[x+5]*mask[1][3]+
ptr[x-
1]*mask[2][1]+ptr[x+2]*mask[2][2]+ptr[x+5]*mask[2][3]+ptr
2[x-
1]*mask[3][1]+ptr2[x+2]*mask[3][2]+ptr2[x+5]*mask[3][3]+.
5);

        x=x+2;
       }
      }
      if(img==1)
      {
       Image1->Canvas->Draw(0,0,pBitmap->Bitmap);
       Image1->Repaint();
       pBitmap->SaveToFile("Img1.bmp");
      }
      if(img==2)
      {
       Image2->Canvas->Draw(0,0,pBitmap->Bitmap);
       Image2->Repaint();
       pBitmap->SaveToFile("Img2.bmp");
      }
      delete pBitmap;
      delete pBitmap2;
    }
   }
}
//---------------------------------------------------------------------------
---

void TForm1::gray_borde_bin(int img)
{
  float a1,a2,a3,b1,b2,b3,f;
  int sector;

  Graphics::TPicture *pBitmap = new Graphics::TPicture();
  Graphics::TPicture *pBitmap2 = new Graphics::TPicture();
  Graphics::TPicture *pBitmap3 = new Graphics::TPicture();
  iniimg(pBitmap,pBitmap2,img);
  //escala de grises
  for (int y = 0; y < pBitmap->Bitmap->Height-1; y++)
  {
    ptr = (Byte *)pBitmap->Bitmap->ScanLine[y];
    for ( int x = 0; x < ((pBitmap->Bitmap->Width)*3)-3; x++)
```

```
    {
     //componentes rgb
     f=0.299*ptr[x+2]+0.587*ptr[x+1]+0.114*ptr[x];
     ptr[x]=floor(f+0.5);
     ptr[x+1]=floor(f+0.5);
     ptr[x+2]=floor(f+0.5); //*/

     x=x+2;
    }
   }
   if(img==1)
   {
    //Lienzo3->Canvas->Draw(0,0,pBitmap->Bitmap);
    pBitmap->SaveToFile("Img1g.bmp");
   }
   if(img==2)
   {
    //Lienzo4->Canvas->Draw(0,0,pBitmap->Bitmap);
    pBitmap->SaveToFile("Img2g.bmp");
   }

   //Sobel (deteccion de bordes) con imagen en gris
   if(BordeCanny1->Checked==false)
   {
    for (int y = 1; y < pBitmap->Bitmap->Height-1; y++)
     {
      ptr0 = (Byte *)pBitmap->Bitmap->ScanLine[y-1];
      ptr1 = (Byte *)pBitmap->Bitmap->ScanLine[y];
      ptr2 = (Byte *)pBitmap->Bitmap->ScanLine[y+1];

      ptr = (Byte *)pBitmap2->Bitmap->ScanLine[y];
      ptr3 =(Byte *)pBitmap2->Bitmap->ScanLine[y-1];
      for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3; x++)
      {
       a1=(ptr0[x-
3]*sx[1][1]+ptr0[x]*sx[1][2]+ptr0[x+3]*sx[1][3]+ptr1[x-
3]*sx[2][1]+ptr1[x]*sx[2][2]+ptr1[x+3]*sx[2][3]+ptr2[x-
3]*sx[3][1]+ptr2[x]*sx[3][2]+ptr2[x+3]*sx[3][3]);
       a2=(ptr0[x-
2]*sx[1][1]+ptr0[x+1]*sx[1][2]+ptr0[x+4]*sx[1][3]+ptr1[x-
2]*sx[2][1]+ptr1[x+1]*sx[2][2]+ptr1[x+4]*sx[2][3]+ptr2[x-
2]*sx[3][1]+ptr2[x+1]*sx[3][2]+ptr2[x+4]*sx[3][3]);
       a3=(ptr0[x-
1]*sx[1][1]+ptr0[x+2]*sx[1][2]+ptr0[x+5]*sx[1][3]+ptr1[x-
1]*sx[2][1]+ptr1[x+2]*sx[2][2]+ptr1[x+5]*sx[2][3]+ptr2[x-
1]*sx[3][1]+ptr2[x+2]*sx[3][2]+ptr2[x+5]*sx[3][3]);

       b1=(ptr0[x-
3]*sy[1][1]+ptr0[x]*sy[1][2]+ptr0[x+3]*sy[1][3]+ptr1[x-
3]*sy[2][1]+ptr1[x]*sy[2][2]+ptr1[x+3]*sy[2][3]+ptr2[x-
3]*sy[3][1]+ptr2[x]*sy[3][2]+ptr2[x+3]*sy[3][3]);
       b2=(ptr0[x-
2]*sy[1][1]+ptr0[x+1]*sy[1][2]+ptr0[x+4]*sy[1][3]+ptr1[x-
2]*sy[2][1]+ptr1[x+1]*sy[2][2]+ptr1[x+4]*sy[2][3]+ptr2[x-
2]*sy[3][1]+ptr2[x+1]*sy[3][2]+ptr2[x+4]*sy[3][3]);
       b3=(ptr0[x-
1]*sy[1][1]+ptr0[x+2]*sy[1][2]+ptr0[x+5]*sy[1][3]+ptr1[x-
1]*sy[2][1]+ptr1[x+2]*sy[2][2]+ptr1[x+5]*sy[2][3]+ptr2[x-
1]*sy[3][1]+ptr2[x+2]*sy[3][2]+ptr2[x+5]*sy[3][3]);

       ptr[x]=sqrt(a1*a1+b1*b1);
       ptr[x+1]=sqrt(a2*a2+b2*b2);
       ptr[x+2]=sqrt(a3*a3+b3*b3);

       //anula cuadro blanco del rededor
       if((x<8)||(x>((pBitmap->Bitmap->Width)*3)-9))//lineas
verticales
       {
        ptr[x-3]=1;
        ptr[x-2]=1;
        ptr[x-1]=1;
        ptr[x]=1;
        ptr[x+1]=1;
        ptr[x+2]=1;
       }
       if(y<5)                    //horizontales del inicio
       {
        ptr3[x]=1;
        ptr3[x+1]=1;
        ptr3[x+2]=1;
       }
       if(y>=pBitmap->Bitmap->Height-3)      //horizontales del
final
       {
        ptr[x]=1;
        ptr[x+1]=1;
        ptr[x+2]=1;
       }

       //realiza binarizacion con threshold


       f=0.299*ptr[x+2]+0.587*ptr[x+1]+0.114*ptr[x];
       if(f<thres)
       {
        ptr[x]=1;
        ptr[x+1]=1;
        ptr[x+2]=1;
       }
       else
       {
        ptr[x]=255;
        ptr[x+1]=255;
        ptr[x+2]=255;
       }
       x=x+2;
      }//fin x
     }//fin y

      //imagen con sobel y binarizada salva a archivo y dibuja
     if(img==1)
     {
      Lienzo1->Canvas->Draw(0,0,pBitmap2->Bitmap);
      pBitmap2->SaveToFile("Img1s.bmp");
     }
     if(img==2)
     {
      Lienzo2->Canvas->Draw(0,0,pBitmap2->Bitmap);
      pBitmap2->SaveToFile("Img2s.bmp");
     }
   }  //termina sobel

    //canny (deteccion de bordes) con imagen en gris
   if(BordeCanny1->Checked==true)
   {


    for (int y = 1;  y< pBitmap->Bitmap->Height-1; y++)
    {
     ptr0 = (Byte *)pBitmap->Bitmap->ScanLine[y-1];
     ptr1 = (Byte *)pBitmap->Bitmap->ScanLine[y];
     ptr2 = (Byte *)pBitmap->Bitmap->ScanLine[y+1];

     ptr = (Byte *)pBitmap2->Bitmap->ScanLine[y];
     for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3; x++)
     {
      //hace canny con mascara
      a1=(ptr1[x+3]-ptr1[x]+ptr2[x+3]-ptr2[x]);
      b1=(ptr1[x]-ptr2[x]+ptr1[x+3]-ptr2[x+3]);
```

```
      ptr[x]=sqrt(a1*a1+b1*b1);
      ptr[x+1]=ptr[x];
      ptr[x+2]=ptr[x];

      if(a1<=0.2)a1=0.2;
      f=atan2(b1,a1);
      //sectoriza
      if((f<=M_PI)&&(f>7*M_PI/8))sector=0;
      if((f<=7*M_PI/8)&&(f>5*M_PI/8))sector=3;
      if((f<=5*M_PI/8)&&(f>3*M_PI/8))sector=2;
      if((f<=2*M_PI/8)&&(f>M_PI/8))sector=1;

      if((f<=M_PI/8)&&(f>-1*M_PI/8))sector=0;
      if((f<=-1*M_PI/8)&&(f>-3*M_PI/8))sector=3;
      if((f<=-3*M_PI/8)&&(f>-5*M_PI/8))sector=2;
      if((f<=-5*M_PI/8)&&(f>-7*M_PI/8)) sector=1;
      if((f<=-7*M_PI/8)&&(f>M_PI)) sector=0;

      ptr1[x]=sector*1;
      ptr1[x+1]=sector*1;
      ptr1[x+2]=sector*1;
      //anula cuadro blanco del rededor
      if((x<8)||(x>((pBitmap->Bitmap->Width)*3)-9))//lineas
verticales
      {
       ptr[x-3]=1;
       ptr[x-2]=1;
       ptr[x-1]=1;
       ptr[x]=1;
       ptr[x+1]=1;
       ptr[x+2]=1;
      }
      if(y<5)                          //horizontales del inicio
      {
       ptr[x]=1;
       ptr[x+1]=1;
       ptr[x+2]=1;
      }
      if(y>=pBitmap->Bitmap->Height-3)      //horizontales del
final
      {
       ptr[x]=1;
       ptr[x+1]=1;
       ptr[x+2]=1;
      }

      x=x+2;
     }//fin x
    }//fin y

    if(img==1)
    {
     Lienzo1->Canvas->Draw(0,0,pBitmap2->Bitmap);
     pBitmap2->SaveToFile("Img1s.bmp");
     pBitmap->SaveToFile("AImg1.bmp");
    }
    if(img==2)
    {
     Lienzo2->Canvas->Draw(0,0,pBitmap2->Bitmap);
     pBitmap2->SaveToFile("Img2s.bmp"); //imagen de bordes
     pBitmap->SaveToFile("AImg2.bmp");  //imagen de
angulos
    }

     //salva a imagenes temporales de canny
     if(img==1)pBitmap2->SaveToFile("ImgC1.bmp");
     if(img==2)pBitmap2->SaveToFile("ImgC2.bmp");
    //realiza binarizacion con threshold pero
   //con los 2 pixeles en línea
    if(img==1)pBitmap3->Bitmap-
>LoadFromFile("Img1s.bmp");
    if(img==2)pBitmap3->Bitmap-
>LoadFromFile("Img2s.bmp");

   for ( int y = 1; y < pBitmap->Bitmap->Height-1; y++)
    {
     ptr0 = (Byte *)pBitmap2->Bitmap->ScanLine[y-1];
//contienen los bordes
     ptr1 = (Byte *)pBitmap2->Bitmap->ScanLine[y];
     ptr2 = (Byte *)pBitmap2->Bitmap->ScanLine[y+1];

     ptr = (Byte *) pBitmap->Bitmap->ScanLine[y];  //contiene
el sector
     ptr3= (Byte *)pBitmap3->Bitmap->ScanLine[y];
//deposita imagen final
      for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-3; x++)
      {
       sector=ptr[x];

       ptr3[x]=1;
       ptr3[x+1]=1;
       ptr3[x+2]=1;
       if(sector==0)
       {
        if((ptr1[x]>ptr1[x-
3])&&(ptr1[x]>ptr1[x+3])&&(ptr1[x]>thres))//si el medio es
mas grande bueno

//if((ptr1[x]>ptr0[x])&&(ptr1[x]>ptr2[x])&&(ptr1[x]>thres))//
si el medio es mas grande
         {
          ptr3[x]=255;
          ptr3[x+1]=255;
          ptr3[x+2]=255;
         }
         else
         {
          ptr3[x]=1;
          ptr3[x+1]=1;
          ptr3[x+2]=1;
         }
       }
       if(sector==1)
       {
        if((ptr1[x]>ptr2[x-
3])&&(ptr1[x]>ptr0[x+3])&&(ptr1[x]>thres))//si el medio es
mas grande
         //if((ptr1[x]>ptr0[x-
3])&&(ptr1[x]>ptr2[x+3])&&(ptr1[x]>thres))//si el medio es
mas grande
         {
          ptr3[x]=255;
          ptr3[x+1]=255;
          ptr3[x+2]=255;
         }
         else
         {
          ptr3[x]=1;
          ptr3[x+1]=1;
          ptr3[x+2]=1;
         }
       }
       if(sector==2)
       {

if(((ptr1[x]>ptr0[x])&&(ptr1[x]>ptr2[x])&&(ptr1[x]>thres))||((
ptr1[x]>ptr1[x-
3])&&(ptr1[x]>ptr1[x+3])&&(ptr1[x]>thres)))//si el medio es
mas grande bueno segundo con truco
```

```
        {
         ptr3[x]=255;
         ptr3[x+1]=255;
         ptr3[x+2]=255;
        }
        else
        {
         ptr3[x]=1;
         ptr3[x+1]=1;
         ptr3[x+2]=1;
        }
       }

       if(sector==3)
       {
        if((ptr1[x]>ptr0[x-
3])&&(ptr1[x]>ptr2[x+3])&&(ptr1[x]>thres))//si el medio es
mas grande
        //if((ptr1[x]>ptr2[x-
3])&&(ptr1[x]>ptr0[x+3])&&(ptr1[x]>thres))//si el medio es
mas grande
        {
         ptr3[x]=255;
         ptr3[x+1]=255;
         ptr3[x+2]=255;
        }
        else
        {
         ptr3[x]=1;
         ptr3[x+1]=1;
         ptr3[x+2]=1;
        }
       }
       x=x+2;
     }//fin x
    }//fin y
    //salva a imagenes de bordes
    if(img==1)
    {
     Lienzo1->Canvas->Draw(0,0,pBitmap3->Bitmap);
     pBitmap3->SaveToFile("Img1s.bmp");
    }
    if(img==2)
    {
     Lienzo2->Canvas->Draw(0,0,pBitmap3->Bitmap);
     pBitmap3->SaveToFile("Img2s.bmp");
    }
  } //finaliza canny

  delete pBitmap;
  delete pBitmap2;
  delete pBitmap3;
}
//---------------------------------------------------------------------------
---
void TForm1::Hough(int img)
{
  Graphics::TPicture *pBitmap = new Graphics::TPicture();
  int cx,cy,x,y,x2,y2;
  int max_rad=600;                    //radio maximo
  float angulo,angulo2,m;
  int r,t,t2;
  int val_div=thres_z*4;              //valor division
  float incr_ang;
  int umbral_car;                     //umbral de no. caracteristicas

  if(img==1){
    //Errors->Lines->Clear();
    pBitmap->Bitmap->LoadFromFile("Img1g.bmp");
    Lienzo3->Canvas->Draw(0,0,pBitmap->Bitmap);
    pBitmap->Bitmap->LoadFromFile("Img1s.bmp");
  }
  if(img==2){
    pBitmap->Bitmap->LoadFromFile("Img2g.bmp");
    Lienzo4->Canvas->Draw(0,0,pBitmap->Bitmap);
    pBitmap->Bitmap->LoadFromFile("Img2s.bmp");
  }

  pBitmap->Bitmap->PixelFormat=pf24bit,pf24bit;

  for(int k=0;k<max_img_y;k++)              //inicializa super
areglo a 0
  {
   for(int l=0;l<max_img_x;l++)
   {
    for(t=0;t<thres_z*2;t++)
    {
     s_arr[0][l][k][t]=false;
     s_arr[1][l][k][t]=false;
    }
   }
  }

  for(t=0;t<361;t++)                    //inicializa areglo a 0
  {
   for(r=0;r<max_rad;r++)
    arreglo[t][r]=0;
  }
  //centro de la imagen
  cy=pBitmap->Bitmap->Height/2;
  cx=pBitmap->Bitmap->Width/2;

  //dibuja ejes no es necesario correr ya lo posiciona
  DrawLinea(0,0,0,-100,0,255,0,pBitmap,(img+2));
  DrawLinea(0,0,100,0,0,255,0,pBitmap,(img+2));
  DrawLinea(0,0,0,100,0,255,0,pBitmap,(img+2));
  DrawLinea(0,0,-100,0,0,255,0,pBitmap,(img+2));

  img--;  //para poner en super arreglo las características
  incr_ang=(2.0*M_PI)/val_div;
  //obtiene caracteristicas y las suma en el areglo
  for (int j = 0; j < pBitmap->Bitmap->Height-1; j++)
  {
   ptr = (Byte *)pBitmap->Bitmap->ScanLine[j];

   x=-cx;
   y=cy-j;
   for (int i = 0; i < ((pBitmap->Bitmap->Width)*3)-3; i++)
   {
    x++;
    if(ptr[i]==255)   //si es uno(característica)
    {
     angulo=0.0;
     for(int ciclo=1;ciclo<=val_div/2;ciclo++)  //ciclo angulo
de .1 a 180
     {
      //angulo=((2.0*M_PI)/val_div)*ciclo;  //angulo del
vector en rad
      angulo+=incr_ang;              //para que las operaciones
sean sumas
      get_pol(x,y,angulo,pBitmap);    //obtiene polares con x,y
cartesianas
     }
    } //si es borde
    i+=2;
   }  // end x
  }  //end y
  //Lienzo3->Repaint();
  //Lienzo4->Repaint();
```

```
  //idea: hacer una segunda pasada donde si el # car es mayor
enonces ponga tambien en s  arr
  umbral_car=( sqrt((pBitmap->Bitmap->Height)*(pBitmap-
>Bitmap->Width)))/thres_h;

  for (int j = 0; j < pBitmap->Bitmap->Height-1; j++)
  {
    ptr = (Byte *)pBitmap->Bitmap->ScanLine[j];
    //Status->SimpleText=" "+FloatToStr(j)+" |
"+((j*100)/(pBitmap->Bitmap->Height))+"%"; //escribe
porcentaje de imagen realizada
    x=-cx;
    y=cy-j;
    for (int i = 0; i < ((pBitmap->Bitmap->Width)*3)-3; i++)
    {
      x++;
      if(ptr[i]==255)   //si es uno(característica)
      {

        angulo=0.0;
        for(int ciclo=1;ciclo<=val_div/2;ciclo++)  //ciclo angulo
de .1 a 180
        {
          //angulo=((2.0*M_PI)/val_div)*ciclo;  //angulo del
vector en rad
          angulo+=incr_ang;

          get_rt(x,y,angulo,pBitmap);//obteine r y t en variables tet
y rad
          //if(ciclo==0)r=abs(y);
          if(arreglo[tet][rad]>umbral_car)
          {
            s_arr[img][(i/3)][j][ciclo-1]=true; //si es mayor al
umbral pone
            //Errors->Lines->Add("imagen:"+IntToStr(img)+"|
X:"+IntToStr(x)+"| Y:"+IntToStr(y)+"|
Angulo:"+IntToStr(((360/val_div)*ciclo)));
            if(PintaBorde1->Checked==true)
            {
              if(img==0)polar(rad,tet,pBitmap,1);
              if(img==1)polar(rad,tet,pBitmap,2);
            }

            //pintado de los detectados quitar para rapidez
            //if(img==0)Lienzo3->Canvas->Pixels[x+cx][cy-
y]=0xff0000;
            //if(img==1)Lienzo4->Canvas->Pixels[x+cx][cy-
y]=0xff0000;
          }
        }
      } //end si es borde
      i+=2;
    }  //end x
    //Lienzo3->Repaint();
    //Lienzo4->Repaint();
  }  //end y


  //Errors->Lines->SaveToFile("out.txt");
  if(img==2){
    //Errors->Lines->Clear();

  }

  delete pBitmap;
}
//---------------------------------------------------------------------------
---
void TForm1::Matching() //obtiene pares de caracteristicas
similares
{
  Graphics::TPicture *pBitmap = new Graphics::TPicture();
  Graphics::TPicture *pBitmap2 = new Graphics::TPicture();

  Graphics::TPicture *pBitmapG = new Graphics::TPicture();
  Graphics::TPicture *pBitmap2G = new Graphics::TPicture();

  int car1[255],car2[255],c1=0,c2=0,j;    //arreglo
características y puntero int menor_arr;
  int menor_carac,result;                 //major y actual match


  iniimg2(pBitmap,pBitmap2);              //carga imagen binaria
en pbitmap y pabitmap2

  pBitmapG->Bitmap->LoadFromFile("Img1g.bmp");
  pBitmap2G->Bitmap->LoadFromFile("Img2g.bmp");

  pBitmapG->Bitmap->PixelFormat=pf24bit,pf24bit;
  pBitmap2G->Bitmap->PixelFormat=pf24bit,pf24bit;

  //inicializacion arreglo de match pairs
  for(int ciclo=0;ciclo<512;ciclo++)
  {
    pares[ciclo][0]=0;
    pares[ciclo][1]=0;
    pares[ciclo][2]=0;
    pares[ciclo][3]=0;

  }
  ptr_pares=0;

  Errors->Lines->Clear();
  for (int y = 1; y < pBitmap->Bitmap->Height-1;
y=y+thres_lin) //recorrido cada y lineas
  //hace recorrido solo sobre una sóla línea
  //for (int y = pBitmap->Bitmap->Height/2; y < (pBitmap-
>Bitmap->Height/2)+1; y++)
  {
    ptr = (Byte *)pBitmap->Bitmap->ScanLine[y];
    ptr2 = (Byte *)pBitmap2->Bitmap->ScanLine[y];

    ptr0 = (Byte *)pBitmapG->Bitmap->ScanLine[y];
    ptr1 = (Byte *)pBitmap2G->Bitmap->ScanLine[y];
    j=0;
    c1=0;
    c2=0;
    //pone en un arreglo las características imagen derecha e izq
    for (int x = 3; x < ((pBitmap->Bitmap->Width)*3)-6; x++)
    {
      j++;
      if(ptr[x]==255)
      {
        car1[c1]=j;//arreglo imagenes izq
        c1++;
        Errors->Lines->Add("Car1:"+IntToStr(j));
      }
      if(ptr2[x]==255)
      {
        car2[c2]=j;//arreglo imagenes der
        c2++;
        Errors->Lines->Add("Car2:"+IntToStr(j));
      }

      x=x+2;
    }
    //Status->SimpleText=IntToStr(c1)+" "+c2+" yes:"+y;
```

```
    if((c1==c2)&&(c1>0)&&(c2>0))//si arreglos tienen igual
longitud
    {
     for(int ciclo1=0;ciclo1<c1;ciclo1++)
     {
      Errors->Lines->Add("xi:"+IntToStr(car1[ciclo1])+"|
yi:"+y+"| xd:"+car2[ciclo1]+"| yd:"+y+"
#pix="+(car1[ciclo1]-car2[ciclo1]));//izquierdo-derecho
     }
    }

    if((c2!=c1)&&(c1>0)&&(c2>0))
    {
    //quita las características no compatibles inclinación y gris
    //asociacion con el mas similar (menor diferencia y menor a
threshold)
     if(c1<c2)  //si la izquierda tiene menos
     {
      for(int ciclo1=0;ciclo1<c1;ciclo1++)
      {
       menor_carac=10000;
       for(int ciclo2=0;ciclo2<c2;ciclo2++)
       {  //ptr[car1[ciclo1],ptr2[car2[ciclo2]]
        result=diferencia(ptr0[((car1[ciclo1])*3)-
3],ptr0[(car1[ciclo1])*3],ptr0[((car1[ciclo1])*3)+3],ptr1[((car2
[ciclo2])*3)-
3],ptr1[(car2[ciclo2])*3],ptr1[((car2[ciclo2])*3)+3],car1[ciclo1
],car2[ciclo2],y,1);
         if(result<menor_carac)
         {
           //asigna a arreglo
          pares[ ptr_pares][0]=car1[ciclo1];
          pares[ ptr_pares][1]=y;
          pares[ ptr_pares][2]=car2[ciclo2];
          pares[ ptr_pares][3]=y;
          menor_carac=result;
         }
       }
       Errors->Lines-
>Add("xi:"+IntToStr(pares[ptr_pares][0])+"| yi:"+y+"|
xd:"+pares[ptr_pares][2]+"| yd:"+y+"
#pix="+(pares[ptr_pares][0]-pares[ ptr_pares][2]));//izquierdo-
derecho
       ptr_pares++;              //incrementa para siguiente match
      }
     }
     else   //(c2<c1)  //si la derecha tiene menos
     {
      for(int ciclo2=0;ciclo2<c2;ciclo2++)
      {
       menor_carac=10000;
       for(int ciclo1=0;ciclo1<c1;ciclo1++)
       {
        result=diferencia(ptr0[((car1[ciclo1])*3)-
3],ptr0[(car1[ciclo1])*3],ptr0[((car1[ciclo1])*3)+3],ptr1[((car2
[ciclo2])*3)-
3],ptr1[(car2[ciclo2])*3],ptr1[((car2[ciclo2])*3)+3],car1[ciclo1
],car2[ciclo2],y,1);
         if(result<menor_carac)
         {
          //asigna a arreglo
          pares[ ptr_pares][0]=car1[ciclo1];
          pares[ ptr_pares][1]=y;
          pares[ ptr_pares][2]=car2[ciclo2];
          pares[ ptr_pares][3]=y;
          menor_carac=result;
         }
       }
       Errors->Lines-
>Add("xi:"+IntToStr(pares[ptr_pares][0])+"| yi:"+y+"|
xd:"+pares[ptr_pares][2]+"| yd:"+y+"
#pix="+(pares[ptr_pares][0]-pares[ ptr_pares][2]));//izquierdo-
derecho
       ptr_pares++;              //incrementa para siguiente match
      }
     }
    } // si ya esta normalizado (cada uno con un par)
  }   //fin y

  dosDtotresD(pBitmap);

  delete pBitmap;
  delete pBitmap2;

  delete pBitmapG;
  delete pBitmap2G;
}
//---------------------------------------------------------------------
---
void __fastcall TForm1::Camara1Click(TObject *Sender)
{
  //intento de configuracion de la camara
  Video->Dialogs;

  //Video->ShowDialog();
  //Video->InsertControl();
  //Video->Controls();
  //Video->GetInterface();
  //Video->
}
//---------------------------------------------------------------------
---
void __fastcall TForm1::Calibracion1Click(TObject *Sender)
{
  FormCalibration=new TFormCalibration (Application);
  FormCalibration->ShowModal();
  delete FormCalibration;
}
//---------------------------------------------------------------------
---
void __fastcall TForm1::Extra3DClick(TObject *Sender)
{
  FormTresD = new TFormTresD (Application);
  FormTresD->ShowModal();
  //FormTresD->Close();
  delete FormTresD;
}
//---------------------------------------------------------------------
-
void __fastcall TForm1::SigmoidalClick(TObject *Sender)
{
  if(Sigmoidal->Checked==true)
  {
    Sigmoidal->Checked=false;
  }
  else
  {
    Sigmoidal->Checked=true;
  }
}
//---------------------------------------------------------------------
-
void __fastcall TForm1::FiltradoONClick(TObject *Sender)
{
  if(FiltradoON->Checked==true)
    FiltradoON->Checked=false;
  else
    FiltradoON->Checked=true;
}
```

```
//---------------------------------------------------------------------------
-
void __fastcall TForm1::GaussianoClick(TObject *Sender)
{
  if(Gaussiano->Checked==true)
    Gaussiano->Checked=false;
  else
    Gaussiano->Checked=true;
}
//---------------------------------------------------------------------------
-
void __fastcall TForm1::BordeCanny1Click(TObject *Sender)
{
  if(BordeCanny1->Checked==true)//inicializacion sobel
  {
    BordeCanny1->Checked=false;

   thres=52;          //threshold binarizacion
   thres_h=5;         //threshold tamano de no muestras para
hough
   thres_z=8;         //threshold zonas de hough
   thres_c=5000;

   //mascara sobel 3x3
   sx[1][1]=-1.0;
   sx[1][2]=0.0;
   sx[1][3]=1.0;
   sx[2][1]=-2.0;
   sx[2][2]=0.0;
   sx[2][3]=2.0;
   sx[3][1]=-1.0;
   sx[3][2]=0.0;
   sx[3][3]=1.0;

   sy[1][1]=1.0;
   sy[1][2]=2.0;
   sy[1][3]=1.0;
   sy[2][1]=0.0;
   sy[2][2]=0.0;
   sy[2][3]=0.0;
   sy[3][1]=-1.0;
   sy[3][2]=-2.0;
   sy[3][3]=-1.0;
  }
  else  //inicializacion cuando corra canny
  {
    BordeCanny1->Checked=true;
                  //anteriormente 20,8,15
   thres=20;          //threshold binarizacion
   thres_h=7;         //threshold tamano de no muestras para
hough
   thres_z=20;         //threshold zonas de hough
   thres_c=5000;

   sx[1][1]=1.0;
   sx[1][2]=0.0;
   sx[2][1]=0.0;
   sx[2][2]=-1.0;

   sy[1][1]=0.0;
   sy[1][2]=-1.0;
   sy[2][1]=1.0;
   sy[2][2]=0.0;
  }
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::PintaBorde1Click(TObject *Sender)
//Dibuja vectores
{
  if(PintaBorde1->Checked==true)
    PintaBorde1->Checked=false;
  else
    PintaBorde1->Checked=true;
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::CaracteristClick(TObject *Sender)
{ //Despliega pantalla con el matching
  if(Caracterist->Checked==true)
  {
    Caracterist->Checked=false;
    Errors->Visible=false;
  }
  else
  {
    Caracterist->Checked=true;
    Errors->Visible=true;
  }
}
//---------------------------------------------------------------------------
-

void __fastcall TForm1::noajusteClick(TObject * Sender)
{
   if(noajuste->Checked==false)
   {
     exponencial->Checked=false;
     polinomio->Checked=false;
     noajuste->Checked=true;
   }

}
//---------------------------------------------------------------------------
-

void __fastcall TForm1::exponencialClick(TObject * Sender)
{
   if(exponencial->Checked==false)
   {
     exponencial->Checked=true;
     polinomio->Checked=false;
     noajuste->Checked=false;
   }
}
//---------------------------------------------------------------------------
-

void __fastcall TForm1::polinomioClick(TObject * Sender)
{
    if(polinomio->Checked==false)
   {
     exponencial->Checked=false;
     polinomio->Checked=true;
     noajuste->Checked=false;
   }
}
//---------------------------------------------------------------------------
-
void __fastcall TForm1::OGLButtonClick(TObject * Sender)
{
  FormTresD = new TFormTresD ( Application);
  FormTresD->ShowModal();
  //FormTresD->Close();
  delete FormTresD;
}
//---------------------------------------------------------------------------
-
void __fastcall TForm1::Salir1Click(TObject * Sender)
{
```

```
  Close();
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::CloseButtonClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------
-
void __fastcall TForm1::FormDestroy(TObject * Sender)
{
  delete config;
}
//---------------------------------------------------------------------------
---
void __fastcall TForm1::FormClose(TObject *Sender,
TCloseAction & Action)
{
  Video->StopPreview();
}
//---------------------------------------------------------------------------
---
//Status->SimpleText=IntToStr(thres)
```