UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

"RECONOCIMIENTO DE CIRCUNFERENCIAS Y LINEAS EN UNA IMAGEN USANDO LA TRANSFORMADA DE HOUGH"

DOCENTE: ING. JOSÉ MAURO PILLCO QUISPE

CURSO: ROBOTICA Y PROCESAMIENTO DE SEÑAL

PRESENTADO POR:

- ACHIRI LAGUNA, ERICK 110128
- CORONEL FERNÁNDEZ BACA, KHAROL 100260
- LOAYZA CALLER, MAYKOL LUDING 101032
- SALAZAR ORÉ, DIEGO MARTÍN 130742
- URQUIZO FUENTES, JESÚS FABRIZIO 101037

Cusco – Perú
2019

ÍNDICE

1. DESCRIPCIÓN DEL PROYECTO

El presente proyecto permite cargar una imagen y analizarla usando la transformada de Hough para el reconocimiento de líneas y circunferencias.

2. MARCO TEÓRICO

2.1. Filtrado de imágenes

El procesamiento digital de imágenes es un tema en el que se está investigando actualmente y se van obteniendo nuevas técnicas que tienen interesantes aplicaciones.

Existen una gran variedad de procedimientos que permiten, a partir de una imagen, obtener otra modificada (técnicas de filtrado). Se trata de métodos con los que se puede resaltar o suprimir, de forma selectiva la información contenida en una imagen, para destacar algunos elementos de ella, o también para ocultar valores anómalos.

Se pueden distinguir entre filtros de paso bajo, de paso alto, direccionales, de detección de bordes, etc. Los filtros de paso bajo tratan de suavizar una imagen, eliminando el posible ruido, o resaltar determinada información presente a una cierta escala. Están basados en la idea de asignar a un pixel el valor en intensidad de color a partir de una ponderación de pixels cercanos. Algunos ejemplos son los filtros de la media, de la media ponderada, de la mediana, adaptativos y gaussianos. Los filtros de paso alto intentan resaltar las zonas de mayor variabilidad, justamente lo contrario que los de paso bajo. Algunos de ellos son los métodos de sustracción de la media y los filtros basados en derivada

2.2. Algoritmo de Canny

Es un operador desarrollado por John F. Canny en 1986 que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes. Se está usando este filtro para la detección de bordes en nuestro programa.

Etapas Algoritmo de Canny

- **Reducción de ruido**

El algoritmo de detección de bordes de Canny utiliza un filtro basado en la primera derivada de una gaussiana. Ya que es susceptible al ruido presente en datos de imagen sin procesar, la imagen original es transformada con un filtro gaussiano. El resultado es una imagen un poco borrosa respecto a la versión original. Esta nueva imagen no se ve afectada por un píxel único de ruido en un grado significativo.

He aquí un ejemplo de un filtro gaussiano 5x5 σ 1.4:

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}$$

- **Encontrar la intensidad del gradiente de la imagen**

El borde de una imagen puede apuntar en diferentes direcciones, por lo que el algoritmo de Canny utiliza cuatro filtros para detectar horizontal, vertical y diagonal en los bordes de la imagen borrosa. El operador de detección de bordes devuelve un valor para la primera derivada en la dirección horizontal (Gy) y la dirección vertical (Gx).

2.3. Transformada de Hough para Líneas

El caso más simple para la transformada de Hough es la transformación lineal para detectar líneas rectas. En el espacio de la imagen, la recta se puede representar con la ecuación:

$$y = m * x + n$$

Y se puede graficar para cada par m,n.

En la transformada de Hough, la idea principal es considerar las características de una recta en término de sus parámetros m y n; y no como puntos de la imagen x,y.

Basándose en lo anterior, una se puede representar como un punto en el espacio de parámetros. Sin embargo, cuando se tienen rectas verticales, los parámetros de la recta m,n se indefinen. Por esta razón es mejor usar los parámetros que describen una recta en coordenada polares, denotados (ρ, θ) y la representacion polar de recta como:

$$\rho = x * cos\theta + y * sin\theta$$

Entonces, es posible asociar a cada recta un par (ρ, θ) que es único si $\theta \in [0, \pi)$ y $\rho \in R$ ó $\theta \in [0, 2 * \pi)$ y $\rho \geq 0$. Este espacio se denomina espacio de Hough para el conjunto de rectas en dos dimensiones.

2.4. Transformada de Hough para Circunferencias

Para la detección de circunferencias se utiliza un sencillo sistema de votación similar al utilizado para la detección de rectas donde sólo habrá que encontrar las casillas más votadas.

La expresión matemática que define una circunferencia es:

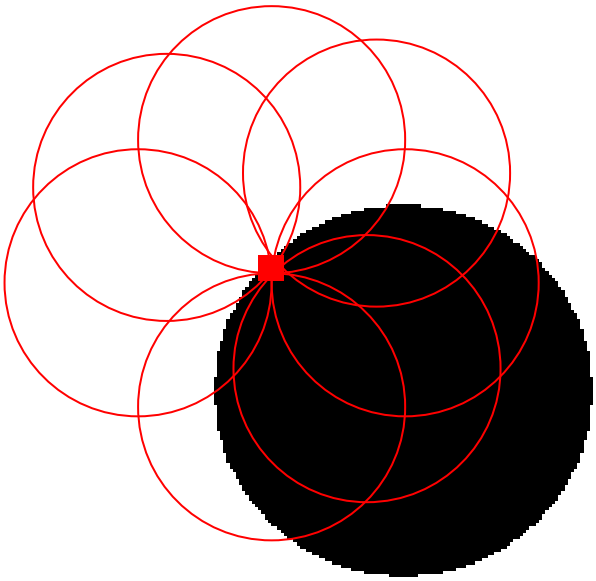
$$\mathbf{(x - c_x)^2 + (y - c_y)^2 = r^2}$$

Este método funciona debido a lo siguiente:

- Valores diferentes de (c_x,c_y,r) proporcionan distintas circunferencias.
- Para cada píxel de contorno que aparece en la posición (x_0, y_0) existe una familia de circunferencias que pasan por este punto dadas por la representación polar:

$$c_x = x_0 + \cos\theta \cdot r$$

$$c_y = y_0 + \sin\theta \cdot r$$



- Cada píxel de contorno vota por todas las circunferencias en la familia (todas las posibles circunferencias que pasan por él).

Pseudo-Codigo

for r=r_min:r_max

for theta=0:360

$c_x=x_0 + \cos(theta)*r;$

$c_y=y_0 + \sin(theta)*r;$

$espacio\_hough(c_x,c_y,r)=espacio\_hough(c_x,c_y,r)+1;$

end

end

3. HERRAMIENTAS UTILIZADA

- Visual studio 2017
- Librería AForge.NET para procesamiento de imágenes.

4. CÓDIGO FUENTE C#

4.1. Clase para análisis usando la transformada de Hough

```
1. using System;
2. using System.Drawing;
3. using System.Collections.Generic;
4. using AForge.Imaging;
5. using AForge.Math.Geometry;
6.
7. namespace lineasCirculosStatico
8. {
9.     public class HoughMap
10.    {
11.        //========= Variable auxiliare Comunes
```

```
12.         //Indices para facilitar busquedas, no tienen importancia e
n el algoritmo
13.         private int iX0 = 0;
14.         private int iX1 = 1;
15.         private int iY0 = 2;
16.         private int iY1 = 3;
17.         private int dXY = 4;
18.         //Parametros estaticos de la imagen, para no realizar llama
da en varios
19.         //metodos
20.         private int ancho = 0;
21.         private int alto = 0;
22.         int centerX = 0;
23.         int centerY = 0;
24.         /// <summary>
25.         /// Imagen resultado con el reconocimiento de lineas y circ
ulos
26.         /// </summary>
27.         public Bitmap imgResultado;
28.         /// <summary>
29.         /// Representacion binaria de la imagen, lo valores de la m
atriz en cero
30.         /// indican que el pixel no tiene informacion para procesar
31.         /// </summary>
32.         private int[,] imgBin;
33.         /// <summary>
34.         /// Espacio de hough para lineas
35.         /// </summary>
36.         public int[,] espacioHoughLinea { get; set; }
37.         /// <summary>
38.         /// Almacena los elementos del espacio de hough para lineas
 con cantidad de votos
39.         /// mayores a la tolerancia definida al procesar la imagen
40.         /// </summary>
41.         private HashSet<Tuple<int, int>> maxLineas;
42.
43.         /// <summary>
44.         /// Almacena 2 puntos con mayor distancia entre si, que rep
resentan
45.         /// una linea de un elemento del espacio de hough para line
as
46.         /// </summary>
47.         private Dictionary<Point, int[]> houghLineas;
48.
49.         /// <summary>
50.         /// Espacio de hough para circunferencias
51.         /// </summary>
```

```
52.         public int[,] espacioHoughCirc { get; set; }
53.         /// <summary>
54.         /// Almacena los elementos del espacio de hough para circun
ferencias con cantidad de votos
55.         /// mayores a la tolerancia definida al procesar la imagen
56.         /// </summary>
57.         private HashSet<Tuple<int, int>> maxCirc;
58.
59.         /// <summary>
60.         ///
61.         /// </summary>
62.         /// <param name="origen">Imagen que se desea procesar</para
m>
63.         public HoughMap(Bitmap origen)
64.         {
65.             imgBin = obtenerMatrizBinaria(origen);
66.             ancho = imgBin.GetLength(0);
67.             alto = imgBin.GetLength(1);
68.             centerX = ancho / 2;
69.             centerY = alto / 2;
70.             imgResultado = new Bitmap(origen);
71.         }
72.         /// <summary>
73.         /// Obtiene la representacion binaria de <paramref name="im
g"/> usada para halla es espacio de hough.
74.         /// </summary>
75.         /// <param name="img">La imagen a procesar ya debe haber pa
sado por algun filtro
76.         /// de deteccion de bordes</param>
77.         /// <returns>Matriz con valores binarios que representa a <
paramref name="img"/> donde los valores
78.         /// en la posicion (x,y) que este en cero indican que el pi
xel (x,y) de <paramref name="img"/> no tiene
79.         /// informacion para procesar</returns>
80.         private int[,] obtenerMatrizBinaria(Bitmap img)
81.         {
82.             int[,] matrizBinaria = new int[img.Width, img.Height];
83.             for (int i = 0; i < img.Width; i++)
84.             {
85.                 for (int j = 0; j < img.Height; j++)
86.                 {
87.                     Color c = img.GetPixel(i, j);
88.                     if (c.R != 0 && c.G != 0 && c.B != 0)
89.                     {
90.                         matrizBinaria[i, j] = 1;
91.                     }
92.                 }
```

```
93.             }
94.             return matrizBinaria;
95.         }
96.         /// <summary>
97.         /// Agrega o modifica al conjunto de lineas encontradas con
 el parametro
98.         /// <paramref name="rhoTheta"/> dependiendo si el punto (<p
aramref name="x"/>,<paramref name="y"/>)
99.         /// genera una linea de mayor longitud que la actual
100.             /// </summary>
101.             /// <param name="rhoTheta">Elemento del espacio de ho
ugh</param>
102.             /// <param name="x">Nueva componenete en x del punto
a analisar</param>
103.             /// <param name="y">Nueva componenete en y del punto
a analisar</param>
104.             private void agregarPuntoLinea(Point rhoTheta, int x,
 int y)
105.             {
106.                 if (houghLineas.ContainsKey(rhoTheta))
107.                 {
108.                     int[] linea = houghLineas[rhoTheta];
109.                     int d0 = distancia(x, y, houghLineas[rhoTheta
][iX0], houghLineas[rhoTheta][iY0]);
110.                     if (linea[dXY] >= 0)
111.                     {
112.                         int d1 = distancia(x, y, houghLineas[rhoT
heta][iX1], houghLineas[rhoTheta][iY1]);
113.                         if (d0 > d1)
114.                         {
115.                             if (d0 > linea[dXY])
116.                             {
117.                                 linea[iX1] = x;
118.                                 linea[iY1] = y;
119.                                 linea[dXY] = d0;
120.                             }
121.                         }
122.                         else
123.                         {
124.                             if (d1 > houghLineas[rhoTheta][dXY])
125.                             {
126.                                 linea[iX0] = x;
127.                                 linea[iY0] = y;
128.                                 linea[dXY] = d1;
129.                             }
130.                         }
131.                     }
```

```
132.                     else
133.                     {
134.                         linea[iX1] = x;
135.                         linea[iY1] = y;
136.                         linea[dXY] = d0;
137.                     }
138.                 }
139.                 else
140.                 {
141.                     int[] aux = new int[5];
142.                     aux[iX0] = x;
143.                     aux[iY0] = y;
144.                     aux[dXY] = -1;
145.                     houghLineas.Add(rhoTheta, aux);
146.                 }
147.             }
148.             /// <summary>
149.             /// Obtiene la distancia entre 2 puntos (<paramref na
me="x0"/>, <paramref name="y0"/>)
150.             /// y (<paramref name="x1"/>,<paramref name="y1"/>)
151.             /// </summary>
152.             /// <param name="x0"></param>
153.             /// <param name="y0"></param>
154.             /// <param name="x1"></param>
155.             /// <param name="y1"></param>
156.             /// <returns></returns>
157.             private int distancia(int x0, int y0, int x1, int y1)
158.             {
159.                 int xV = x0 - x1;
160.                 int yV = y0 - y1;
161.                 return xV * xV + yV * yV;
162.             }
163.
164.             /// <summary>
165.             /// Aplica metodo de Hough para detectar lineas.
166.             /// </summary>
167.             /// <param name="tolerancia">Minimo de puntos coincid
entes en un linea</param>
168.             public void ProcesarLineas(int tolerancia)
169.             {
170.                 int maxTheta = 180;
171.                 int maxRho = (int)(Math.Sqrt(ancho * ancho + alto
 * alto) + 1);
172.                 int houghHeight = (int)(Math.Sqrt(2) * Math.Max(a
ncho, alto)) / 2;
173.                 int doubleHeight = houghHeight * 2;
174.                 int houghHeightHalf = houghHeight / 2;
```

```
175.                 int houghWidthHalf = maxTheta / 2;
176.
177.                 espacioHoughLinea = new int[maxRho, maxTheta];
178.
179.                 // Escaneamos cada pixel de la imagen--+
180.                 for (int y = 0; y < alto; y++)                  //
|
181.                 {
//|
182.                     for (int x = 0; x < ancho; x++)//<-----------
--+
183.                     {
184.                         if (imgBin[x, y] != 0)//Si el el pixel es
 negro lo ignoramos.
185.                         {
186.                             // Determinamos el espacio de hough p
ara lineas.
187.                             // Puede variar de -90 a 90 grados.
188.                             for (int theta = 0; theta < maxTheta;
 theta++)
189.                             {
190.                                 /// Calcula el valor rho para los
 parametros establecidos
191.                                 /// de acuero a la representacion
 polar de la linea
192.                                 double rad = theta * Math.PI / 18
0;
193.                                 double sin = ((x - centerX) * Mat
h.Cos(rad));
194.                                 double cos = ((y - centerY) * Mat
h.Sin(rad));
195.                                 int rho = (int)(sin + cos);
196.
197.                                 // get rid of negative value
198.                                 rho += houghHeight;
199.
200.                                 // Si el valor del radio esta ent
re 1 y el double del alto
201.                                 if ((rho > 0) && (rho <= maxRho))
202.                                 {
203.                                     agregarPuntoLinea(new Point(r
ho, theta), x, y);
204.                                     espacioHoughLinea[rho, theta]
++;
205.                                     //Si existen una cantidad de
puntos mayores a la tolerancia establecida
```

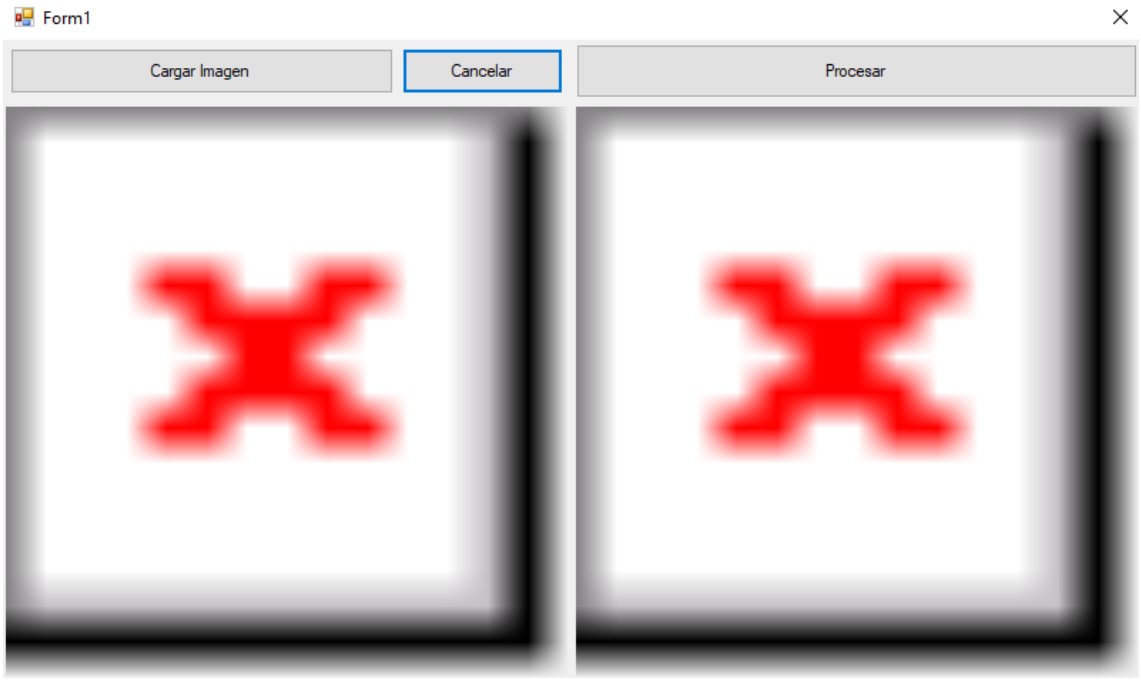
```
206.                                     //se agrega a la lista de lin
eas a dibujar
207.                                     if (espacioHoughLinea[rho, th
eta] > tolerancia)
208.                                     {
209.                                         Tuple<int, int> aux = new
 Tuple<int, int>(rho, theta);
210.                                         maxLineas.Add(aux);
211.                                     }
212.                                 }
213.                             }
214.                         }
215.                     }
216.                 }
217.                 dibujarLineas();
218.             }
219.
220.             public void Procesar(int tolerancia)
221.             {
222.                 if (imgBin != null)
223.                 {
224.                     //Inicializamos los espacios de hough y los m
as botados
225.                     houghLineas = new Dictionary<Point, int[]>();
226.                     maxLineas = new HashSet<Tuple<int, int>>();
227.                     //Procesamos para lineas
228.                     ProcesarLineas(tolerancia);
229.                     //Procesamos para circunferencias
230.                     ProcesarCir();
231.                 }
232.             }
233.
234.             public void ProcesarCir()
235.             {
236.                 // locate objects using blob counter
237.                 BlobCounter blobCounter = new BlobCounter();
238.                 blobCounter.ProcessImage(imgResultado);
239.                 Blob[] blobs = blobCounter.GetObjectsInformation(
);
240.                 // create Graphics object to draw on the image an
d a pen
241.                 Graphics g = Graphics.FromImage(imgResultado);
242.                 Pen pPen = new Pen(Color.Green, 4);
243.                 // check each object and draw circle around objec
ts, which
244.                 SimpleShapeChecker shapeChecker = new SimpleShape
Checker();
```

```
245.                 // are recognized as circles
246.                 for (int i = 0, n = blobs.Length; i < n; i++)
247.                 {
248.                     List<AForge.IntPoint> edgePoints = blobCounte
r.GetBlobsEdgePoints(blobs[i]);
249.
250.                     AForge.Point center;
251.                     float radius;
252.
253.                     if (shapeChecker.IsCircle(edgePoints, out cen
ter, out radius))
254.                     {
255.                         g.DrawEllipse(pPen,
256.                             (int)(center.X - radius),
257.                             (int)(center.Y - radius),
258.                             (int)(radius * 2),
259.                             (int)(radius * 2));
260.                     }
261.                 }
262.
263.                 pPen.Dispose();
264.                 g.Dispose();
265.             }
266.             /// <summary>
267.             /// Accion a tomar cuando se encuentre un pixel parat
e de una linea
268.             /// </summary>
269.             /// <param name="x">Componenete en x del pixel a anal
isar</param>
270.             /// <param name="y">Componenete en y del pixel a anal
isar</param>
271.             /// <returns></returns>
272.             public bool plotLinea(int x, int y)
273.             {
274.                 imgResultado.SetPixel(x, y, Color.Red);
275.                 return true;
276.             }
277.             /// <summary>
278.             /// Funcion a que dibuja las lineas encontradas en la
 imagen de resultado.
279.             /// </summary>
280.             private void dibujarLineas()
281.             {
282.                 foreach (Tuple<int, int> rhoTheta in maxLineas)
283.                 {
284.                     Point linea = new Point(rhoTheta.Item1, rhoTh
eta.Item2);
```

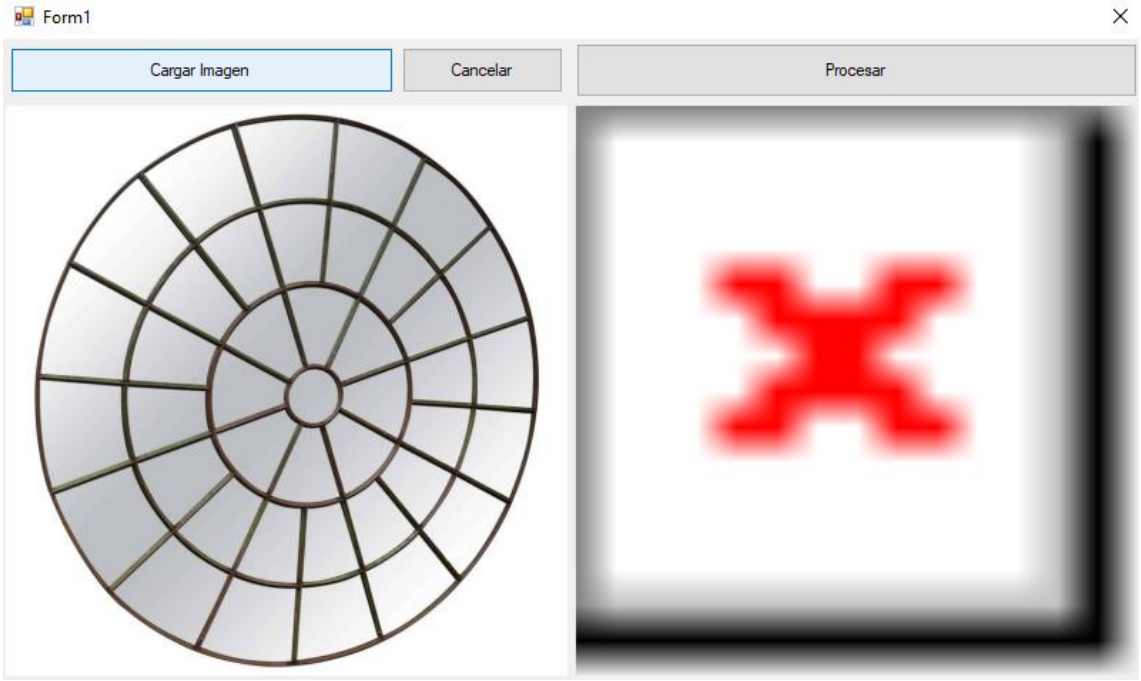
```
285.                     Bresenham.Line(houghLineas[linea][iX0], hough
Lineas[linea][iY0], houghLineas[linea][iX1], houghLineas[linea][iY1],
plotLinea);
286.                 }
287.             }
288.             /// <summary>
289.             /// Aplica metodo de Hough para detectar lineas y cir
cunferencias.
290.             /// </summary>
291.             /// <param name="tolerancia">Minimo de puntos coincid
entes en un linea</param>
292.
293.         }
294.     }
295.
```

5. FOTOGRAFÍAS DE USO

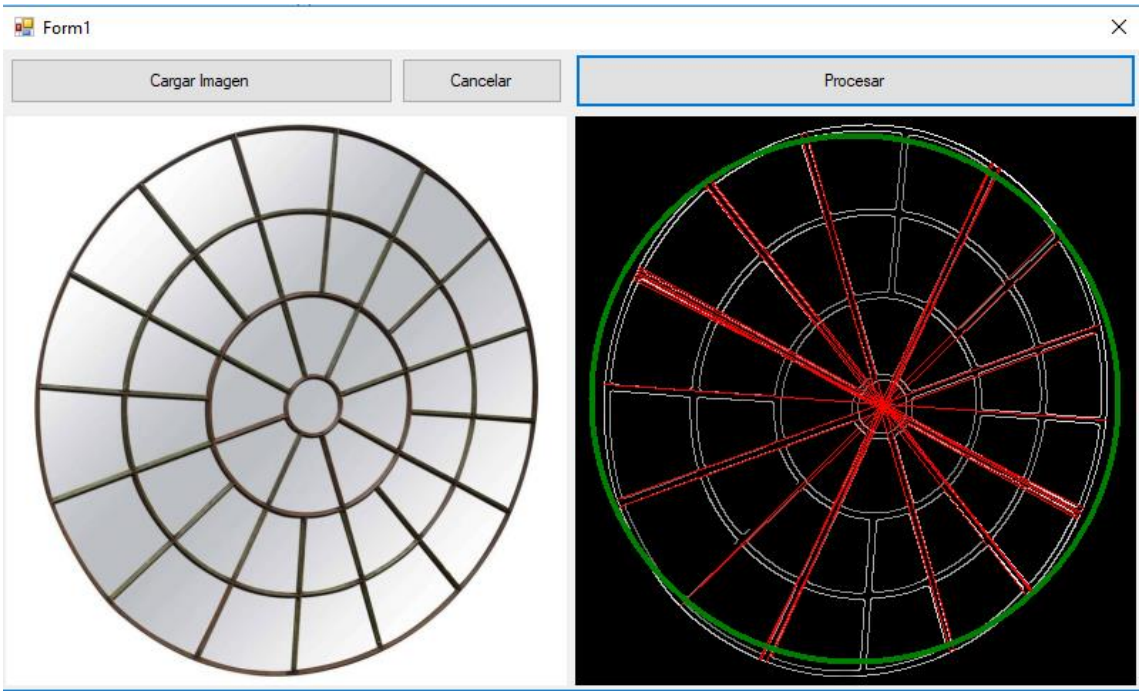
- **PANTALLA DE INICIO**



- **CARGAR IMAGEN**



- **PROCESAR LA IMAGEN**



6. DESCARGA DEL PROYECTO

https://github.com/jesus101037/Robotica-2019-1/tree/master/reconocimiento

La carpeta de este proyecto es “lineaCirculoEstatico”

De ser necesario los dlls de AForge están también en el repositorio, si hay algún problema simplemente actualizar la ubicación de las referencias.