

# **UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO**

**ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE  
SISTEMAS**



## **“CONVOLUCIÓN”**

**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    | 2 |
| 2.   | MARCO teórico               | 2 |
| 2.1. | Filtrado de imágenes        | 2 |
| 2.2. | Convolución                 | 2 |
| 3.   | HERRAMIENTAS UTILIZADA      | 3 |
| 4.   | CÓDIGO FUENTE C#            | 3 |
| 236. | FOTOGRAFÍAS DEL LABORATORIO | 8 |

## 1. DESCRIPCIÓN DEL PROYECTO

El presente proyecto permite cargar una imagen y aplicarle diferentes filtros de convolución.

## 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 píxel el valor en intensidad de color a partir de una ponderación de píxeles 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. Convolución

Es el tratamiento de una matriz por otra que se llama “kernel”.

El filtro matriz de convolución usa una primera matriz que es la imagen que será tratada. La imagen es una colección bidimensional de píxeles en coordenada rectangular. El kernel usado depende del efecto deseado.

El filtro examina, sucesivamente, cada píxel de la imagen. Para cada uno de ellos, que llamaremos “píxeles iniciales”, se multiplica el valor de este píxel y el valor de los 8 circundantes por el valor correspondiente del kernel. Entonces se añade el resultado, y el píxel inicial se regula en este valor resultante final.

Un ejemplo simple:

|    |    |    |    |    |
|----|----|----|----|----|
| 35 | 40 | 41 | 45 | 50 |
| 40 | 40 | 42 | 46 | 52 |
| 42 | 46 | 50 | 55 | 55 |
| 48 | 52 | 56 | 58 | 60 |
| 56 | 60 | 65 | 70 | 75 |

 $\times$ 

|  |   |   |   |  |
|--|---|---|---|--|
|  |   |   |   |  |
|  | 0 | 1 | 0 |  |
|  | 0 | 0 | 0 |  |
|  | 0 | 0 | 0 |  |
|  |   |   |   |  |

 $=$ 

|  |  |    |  |  |
|--|--|----|--|--|
|  |  |    |  |  |
|  |  |    |  |  |
|  |  | 42 |  |  |
|  |  |    |  |  |
|  |  |    |  |  |

A la izquierda, la imagen de la matriz: cada píxel está marcado con su valor. El píxel inicial tiene un borde rojo. El área de acción del kernel tiene un borde verde. En el medio, el kernel, y a la derecha, el resultado de convolución.

Lo que sucede: el filtro lee sucesivamente, de izquierda a derecha y de arriba a abajo, todos los píxeles del área de acción del kernel. Se multiplica el valor de cada uno de ellos por el valor correspondiente del kernel y se suman los resultados:

$$(100 * 0) + (50 * 1) + (50 * 0) + (100 * 0) + (100 * 0) + (100 * 0) + (100 * 0) + (100 * 0) + (100 * 0) + (100 * 0) = 50.$$

El píxel inicial asumió el valor 50. Previamente, cuando el píxel inicial tenía el valor=50, tomó el valor 100 del píxel de arriba (el filtro no trabaja sobre la imagen sino sobre una copia) y de esta manera desapareció en el fondo de píxeles a "100". Como resultado gráfico, el píxel inicial se movió un píxel hacia abajo.

### 3. HERRAMIENTAS UTILIZADA

- Visual studio 2017

### 4. CÓDIGO FUENTE C#

```

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Linq;
7. using System.Text;
8. using System.Threading.Tasks;
9. using System.Windows.Forms;
10.
11. namespace convolucion
12. {
13.     public partial class convInput : Form
14.     {
15.         /// <summary>
16.         /// Tamaño de la matriz de filtro
17.         /// </summary>
18.         int filtroN = 3;
19.
20.         public convInput()
21.         {
22.             InitializeComponent();
23.             inicializar();
24.         }
25.
26.         public void inicializar() {
27.             filtroN = 3;
28.             cbFiltroN.SelectedIndex = 0;
29.             pbOrigen.Image = pbOrigen.ErrorImage;
30.             pbFiltro.Image = pbFiltro.ErrorImage;
31.             inicializarFiltro();
32.         }
33.         public void inicializarFiltro() {
34.             for (int i = 0; i < filtroN; i++)
35.             {

```

```

36.         for (int j = 0; j < filtroN; j++)
37.         {
38.             dgvFiltro[i, j].Value = 0;
39.         }
40.     }
41. }
42.
43.     /// <summary>
44.     /// Evalua que <paramref name="Valor"/> esta en el rango [0;255]
45.     /// </summary>
46.     /// <param name="Valor">Valor a evaluar</param>
47.     /// <returns></returns>
48.     public int analisisValor(int Valor) {
49.         if (Valor < 0)
50.         {
51.             Valor = 0;
52.         }
53.         else if (Valor > 255)
54.         {
55.             Valor = 255;
56.         }
57.         return Valor;
58.     }
59.
60.     /// <summary>
61.     /// Aplica el operador de convolucion para un pixel en la posción
62.     central de <paramref name="subMatriz"/>
63.     con un filtro definido en <paramref name="filtro"/>.
64.     /// </summary>
65.     /// <param name="subMatriz">Sub matriz de la imagen del mismo
66.     tamaño que <paramref name="filtro"/></param>
67.     /// <param name="filtro"></param>
68.     /// <returns>Resultado de aplicar la convolucion entre <paramref
69.     name="subMatriz"/> y <paramref name="filtro"/></returns>
70.     public int convolucionPixel(int[,] subMatriz, double[,] filtro) {
71.         int resultado = 0;
72.         double parcial = 0;
73.         int longitud = subMatriz.GetLength(0);
74.         for (int i = 0; i < longitud; i++)
75.         {
76.             for (int j = 0; j < longitud; j++)
77.             {
78.                 parcial = parcial + subMatriz[i, j] * filtro[i, j];
79.             }
80.         }
81.         resultado = Convert.ToInt32(parcial);
82.         return analisisValor(resultado);
83.     }
84.     /// <summary>
85.     /// Obtienen una matriz de <paramref name="n"/>x<paramref
86.     name="n"/> que rodea al elemento
87.     /// pivot en la posicion (<paramref name="x"/>, <paramref
88.     name="y"/>) de <paramref name="matriz"/>
89.     /// </summary>
90.     /// <param name="matriz">Matriz origen</param>
91.     /// <param name="x">Coordenada en x del pivot</param>
92.     /// <param name="y">Coordenada en y del pivot</param>
93.     /// <param name="n">Tamaño de la matriz cuadrada, debe ser
94.     impar</param>
95.     /// <returns></returns>

```

```

90.         public int[,] obtenerSubmatriz(int[,] matriz, int x, int y, int n)
91.         {
92.             int[,] subMatriz = new int[n, n];
93.             int r = (n / 2);
94.             int maxX = matriz.GetLength(0);
95.             int maxY = matriz.GetLength(1);
96.             int xmin = 0;
97.             int ymin = 0;
98.             int xmax = 0;
99.             int ymax = 0;
100.            for (int i = 0; i < n; i++)
101.            {
102.                xmin = x + i - r;
103.                xmax = x + i + r;
104.                if (xmin >= 0 && xmax < maxX)
105.                {
106.                    for (int j = 0; j < n; j++)
107.                    {
108.                        ymin = y + j - r;
109.                        ymax = y + j + r;
110.                        if (ymin >= 0 && ymax < maxY)
111.                        {
112.                            subMatriz[i, j] = matriz[xmin, ymin];
113.                        }
114.                    }
115.                }
116.            }
117.            return subMatriz;
118.        }
119.        /// <summary>
120.        /// Aplica la operacion de convolucion a cada pixel de cada
121.        canal RGB de la imagen
122.        /// <paramref name="img"/> con el filtro <paramref
123.        name="filtro"/>
124.        /// </summary>
125.        /// <param name="img">Imagen a la cual se desea aplicar el
126.        filtro</param>
127.        /// <param name="filtro">Filtro a aplicar a la imagen, debe
128.        ser una matriz impar</param>
129.        /// <returns></returns>
130.        public Bitmap aplicarFiltro(Bitmap img, double[,] filtro) {
131.            int n = filtro.GetLength(0);
132.            Bitmap imgR = new Bitmap(img.Width, img.Height);
133.            // Variables que contienen la posicion del pivot
134.            //int x, y;
135.            // Matrices q representa la imagen original en sistema
136.            RGB
137.            int[,] imaR = new int[img.Width, img.Height];
138.            int[,] imaG = new int[img.Width, img.Height];
139.            int[,] imaB = new int[img.Width, img.Height];
140.            // Matrices que contendra la imagen en sistema RGB
141.            despues de aplicar el filtro
142.            int[,] imaRR = new int[img.Width, img.Height];
143.            int[,] imaRG = new int[img.Width, img.Height];
144.            int[,] imaRB = new int[img.Width, img.Height];
145.            // Generamos las matrices con los valores de los
146.            canales RGB de la imagen.
147.            for (int x = 0; x < img.Width; x++)
148.            {

```

```

142.         for (int y = 0; y < img.Height; y++)
143.         {
144.             Color oc = img.GetPixel(x, y);
145.             imaR[x, y] = (int)(oc.R);
146.             imaG[x, y] = (int)(oc.G);
147.             imaB[x, y] = (int)(oc.B);
148.         }
149.     }
150.
151.     for (int x = 0; x < img.Width - 1; x++)
152.     {
153.
154.         for (int y = 0; y < img.Height - 1; y++)
155.         {
156.
157.             imaRR[x, y] =
convolucionPixel(obtenerSubmatriz(imaR, x, y, n), filtro);
158.             imaRG[x, y] =
convolucionPixel(obtenerSubmatriz(imaG, x, y, n), filtro);
159.             imaRB[x, y] =
convolucionPixel(obtenerSubmatriz(imaB, x, y, n), filtro);
160.
161.             Color oc = img.GetPixel(x, y);
162.
163.             int colorR = analisisValor(imaRR[x, y]);
164.             int colorG = analisisValor(imaRG[x, y]);
165.             int colorB = analisisValor(imaRB[x, y]);
166.             Color nc = Color.FromArgb(oc.A, colorR, colorG,
colorB);
167.
168.             imgR.SetPixel(x, y, nc);
169.         }
170.     }
171.     return imgR;
172. }
173.
174. public double[,] obtenerFiltro() {
175.     double[,] filtro = new double[filtroN, filtroN];
176.     for (int i = 0; i < filtroN; i++)
177.     {
178.         for (int j = 0; j < filtroN; j++)
179.         {
180.             string aux = dgvFiltro[j, i].Value.ToString();
181.             filtro[i, j] = double.Parse(aux);
182.         }
183.     }
184.     return filtro;
185. }
186.
187. private void buscarImg_click(object sender, EventArgs e)
188. {
189.     OpenFileDialog dialog = new OpenFileDialog();
190.     dialog.Filter = "Image Files|*.jpg;*.jpeg;*.png;...";
191.     dialog.InitialDirectory = @"C:\\";
192.     dialog.Title = "Seleccione image.";
193.     dialog.InitialDirectory =
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
194.
195.     if (dialog.ShowDialog() == DialogResult.OK)
196.     {

```

```

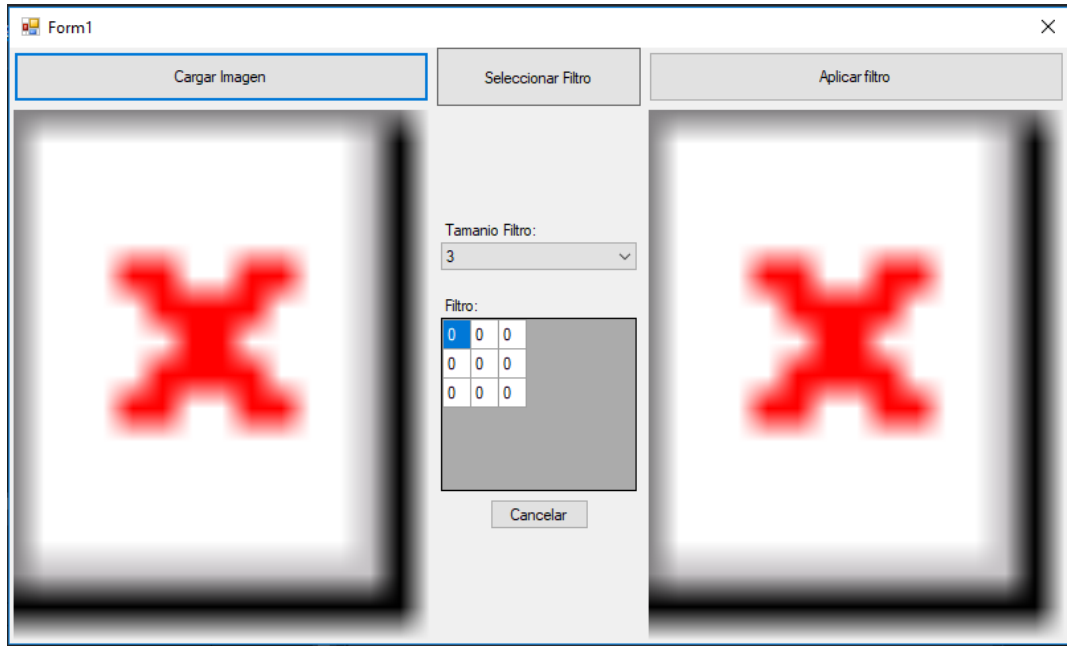
197.                // Recuperar imagen.
198.                Bitmap image = new Bitmap(new
Bitmap(dialog.FileName),pbOrigen.Width, pbOrigen.Height);
199.                pbOrigen.Image = image;
200.            }
201.        }
202.
203.        private void aplicarFiltro_click(object sender, EventArgs
e)
204.        {
205.            //pictureBox2 = new PictureBox();
206.            Bitmap im = (Bitmap)(pbOrigen.Image);
207.            pbFiltro.Image = aplicarFiltro(im, obtenerFiltro());
208.        }
209.
210.        private void buCancelar_Click(object sender, EventArgs e)
211.        {
212.            inicializar();
213.        }
214.
215.        private void cbFiltroN_SelectedIndexChanged(object sender,
EventArgs e)
216.        {
217.            filtroN =
int.Parse(cbFiltroN.Items[cbFiltroN.SelectedIndex].ToString());
218.            dgvFiltro.ColumnCount = filtroN;
219.            dgvFiltro.RowCount = filtroN;
220.            inicializarFiltro();
221.        }
222.    }
223. }

```

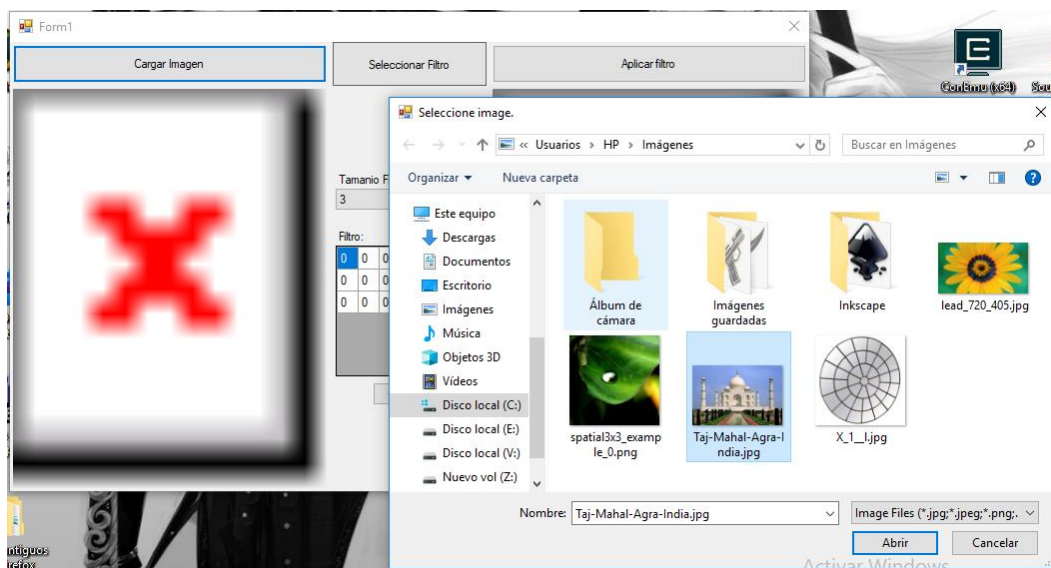


## 5. FOTOGRAFÍAS DE USO

- PANTALLA DE INICIO




- CARGAR IMAGEN



Form1

Cargar Imagen



Seleccionar Filtro

Tamaño Filtro:


3

Filtro:

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Cancelar


Aplicar filtro



- **SELECCIONAMOS EL TAMAÑO DEL FILTRO**

Form1

Cargar Imagen



Seleccionar Filtro

Tamaño Filtro:

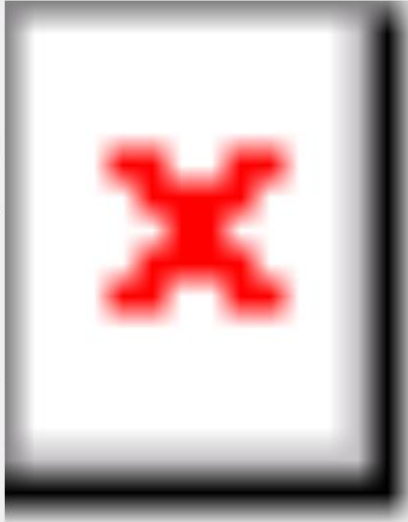
5

Filtro:

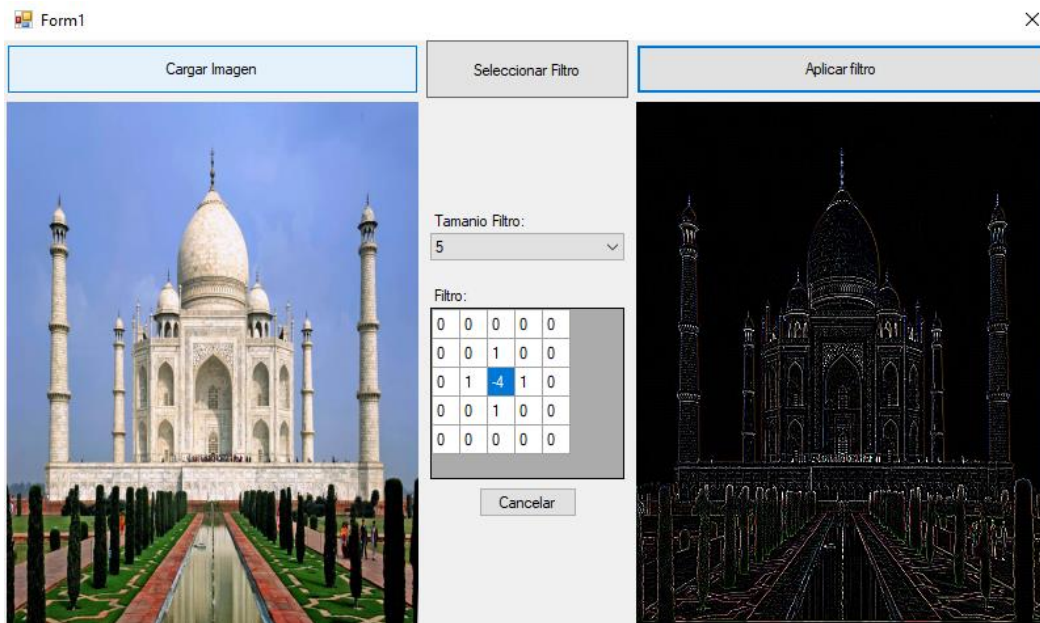
|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Cancelar

Aplicar filtro



- **APLICAR FILTRO**



## 6. DESCARGA DEL PROYECTO

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