PROCESAMIENTO DE IMÁGENES DIGITALES

Angelo Salazar
Juan José Revelo
Juan Jose Bailon

PROCESAMIENTO DE IMÁGENES

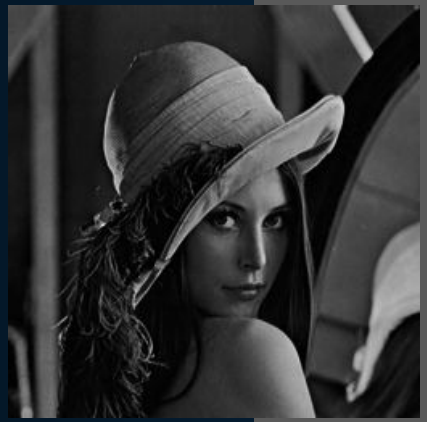
El procesamiento digital de imágenes es uno de los principales objetos de estudio de la actualidad el cual es un campo abierto de la investigación vinculando áreas como la computación, y las matemáticas se plantea y desarrolla un programa orientado objetos el cual permite trabajar con imágenes de formato PPM y PGM en la cual haciendo uso del lenguaje pueden ser modificadas y estudiadas.

¿Qué es el formato PPM?

El formato Portable PixMap (PPM) con colores RGB desde 0 a 225, es un formato sin compresión que puede ser almacenado en formato de texto, por lo que no es necesario realizar un tratamiento binario del fichero.

1 Pixel esta compuesto por 3 valores Red, Green, **Blue**.

¿Qué es el formato PGM?



El formato Portable Grey Map (PGM) puede establecerse desde negro 0 a 255 siendo blanco. Son archivos portables que contienen datos de imagen en forma de mapas de escala de grises, al igual que el PPM puede ser almacenado en un archivo de texto sin comprensión.

P3 *Formato de Imagen PPM*

P2 *Formato de Imagen PGM*

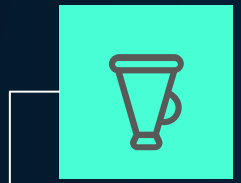
```
# Created by Mr.Repo
11 5
255
92 93 165 156 159 223 219 220 135 123 123
92 92 156 156 156 219 219 219 123 123 123
92 92 156 156 156 219 219 219 123 123 123
92 92 156 156 156 219 219 219 123 123 123
92 92 156 156 156 219 219 219 127 123 123
237 28 36 255 127 39 255 127 39 255 127 39 255 242 0
255 242 0 255 242 0 41 179 82 34 177 76 34 177 76
```

Comentarios identificados con #

Tamaño de Imagen Columnas/Filas

Valor máximo (número mágico)

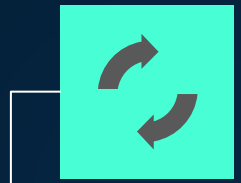
OBJETIVOS



01

FILTRADO

Obtener, a partir de una imagen origen, una nueva cuyo resultado sea más adecuado para una aplicación específica, realizando ciertas modificaciones de la misma.



02

COMBINACIÓN

Crear una composición con imágenes donde se eliminan su fondo para formar una sola y sobreponer una imagen sobre otra.

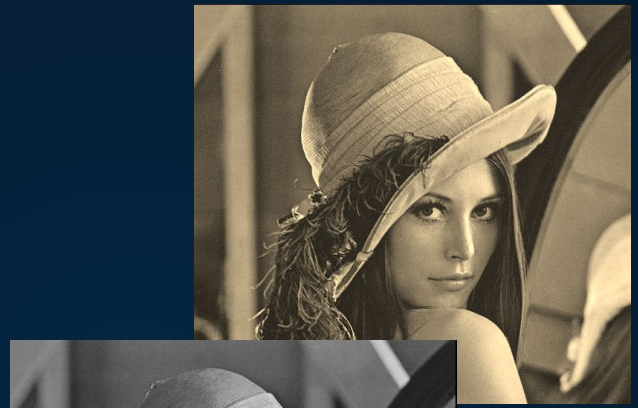
03

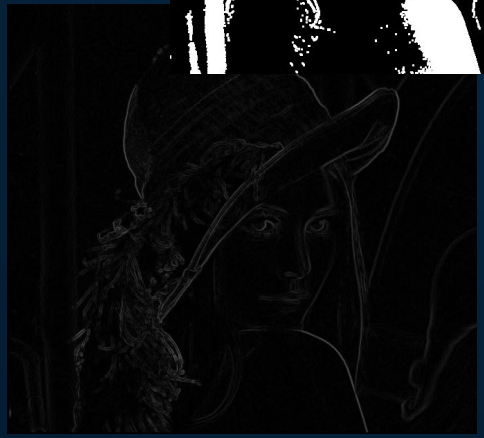
DISPARIDAD BINOCULAR

Determinar profundidad y relieve mediante el uso de cálculos para la posible creación de imágenes 3D.

FILTROS

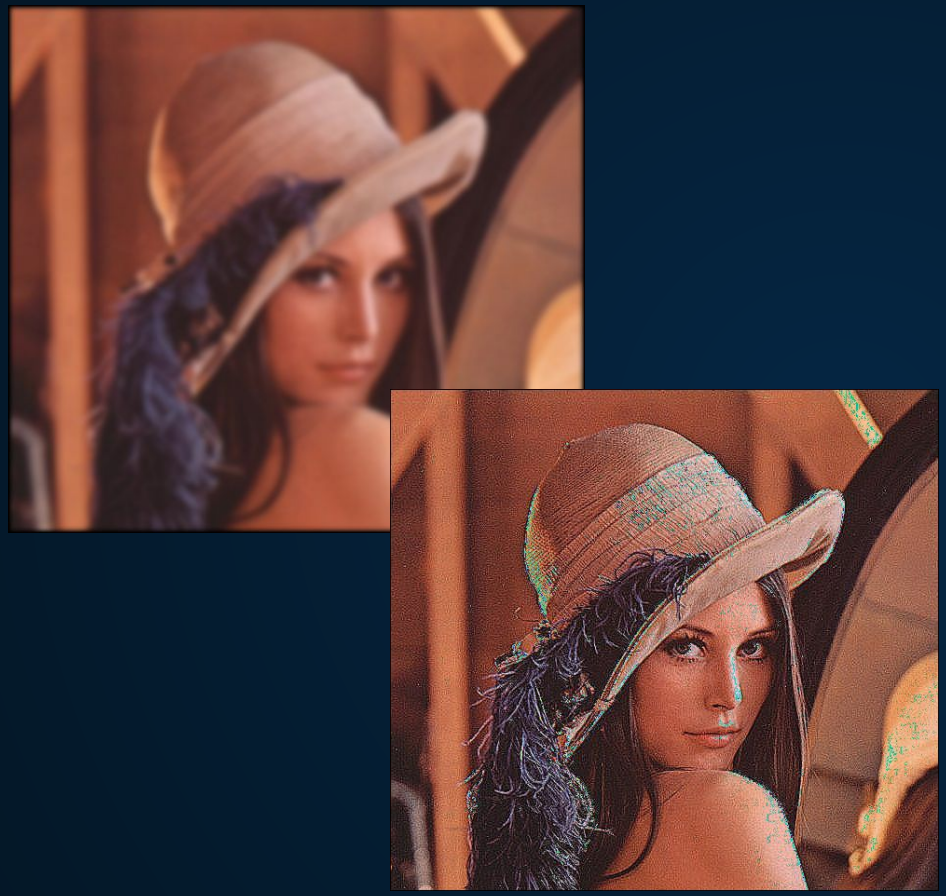
- SEPIA
- GRIS
- FLIPPED AND FLOPPED
- NEGATIVO





FILTROS

- COLORIZE
- THRESHOLD
- EROSIÓN
- DILATACIÓN
- SOBEL



FILTROS

MEAN BLUR

SHARPEN

SEPIA

El sepia es un filtro de tonos marrones cálidos que puede ser aplicado a cualquier imagen teniendo en cuenta el siguiente algoritmo:

```
int tr = round( 0.393*red[i] + 0.769*green[i] + 0.189*blue[i] );
int tg = round( 0.349*red[i] + 0.686*green[i] + 0.168*blue[i] );
int tb = round( 0.272*red[i] + 0.534*green[i] + 0.131*blue[i] );
      unsigned int r = (tr>255) ? 255 : tr;
      unsigned int g = (tg>255) ? 255 : tg;
      unsigned int b = (tb>255) ? 255 : tb;
```

GRIS

El gris es un filtro de tonos grises que permite tener una imagen PPM en esta escala, puede ser aplicado a cualquier imagen teniendo en cuenta el siguiente algoritmo:

```
multiplyVectorBy(0.299, red);
multiplyVectorBy(0.587, green);
multiplyVectorBy(0.114, blue);
  for(int i=0; i<red.size(); i++){
    unsigned int temp = red[i]+green[i]+blue[i];
    red[i]= temp;
    green[i]= temp;
    blue[i]= temp;
  }
```

FLIPPED AND FLOPPED

0	1	2	3	4	5	6	7	8	
1	2	3	4	5	6	7	8	9	1
10	11	12	13	14	15	16	17	18	2
9	10	11	12	13	14	15	16	17	

```
for(int i=rows; i>0; i--){
 for(int i=0; i<rows; i++){
   for(int j=0; j<cols; j++){
     for(int j=0; j<cols; j++){


        tempRed.push_back( red[ (cols*(i-1)) + j ] );
         tempRed.push_back( red[ (cols*(i+1)) - 1 - j ] );
        tempGreen.push_back( green[ (cols*(i-1)) + j ] );
         tempGreen.push_back( green[ (cols*(i+1)) - 1 - j ] );
        tempBlue.push_back( blue[ (cols*(i-1)) + j ] );
         tempBlue.push_back( blue[ (cols*(i+1)) - 1 - j ] );
    } }
} }
```

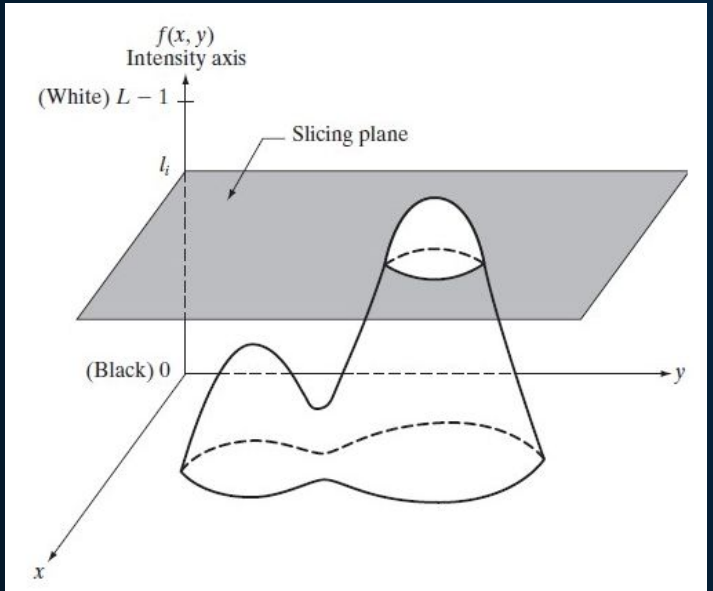
NEGATIVO

El filtro negativo es determinado a partir de calcular el inverso de cada pixel en la imagen, en este caso se debe tener presente o conocer el valor máximo que puede tomar el mismo conocido también como magic number.

```
for(int i = 0; i < red.size(); i++){
        red[i] = maxVal - red[i];
        green[i] = maxVal - green[i];
        blue[i] = maxVal - blue[i];
    }
```

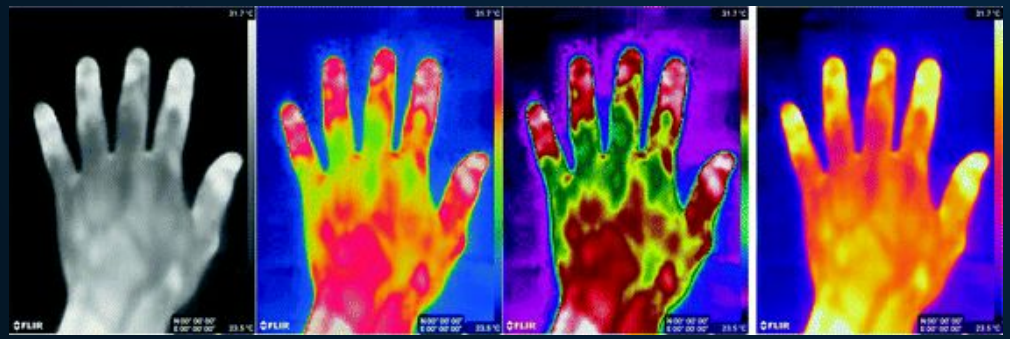
COLORIZE

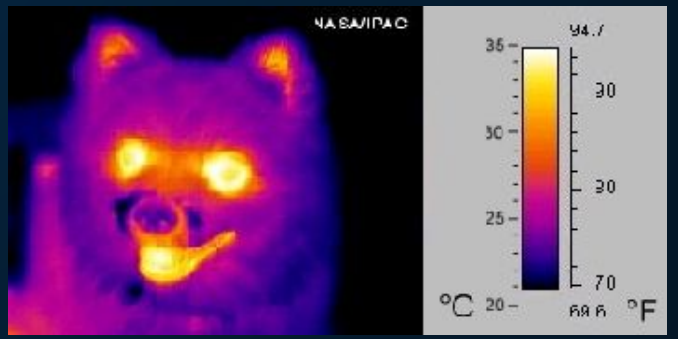
Para la asignacion de falso color a una imagen en blanco y negro, se utilizó una técnica de pseudocolor conocida como Intensity Slicning Normalizada.



COLORIZE

La asignación de los colores se realizó basándose en las intensidades del espectro de luz visible. Se crearon 14 planos por lo tanto existen 13 intervalos de color.





THRESHOLD

Para el filtro Threshold se hizo uso del método Isodata.

Isodata (Iterative Self-Organizing Data Analysis) es una técnica simple iterativa desarrollada por Ridler y Calvard en 1978. El objetivo del algoritmo Isodata consiste en dividir el histograma en dos partes que representan a su vez dos sub-regiones en la imagen. Se trata de un método iterativo que obtiene el umbral mediante los siguientes pasos:

1. Se debe elegir un valor inicial de threshold T.
2. Se debe dividir la imagen en dos grupos R1 y R2 usando T.
3. Se determinan μ1 y μ2

$$\mu 1(T) = \sum_{i=0}^{T} i * h(i) / \sum_{i=0}^{T} h(i)$$

$$\mu 2(T) = \sum_{i=T+1}^{maxValue} i * h(i) / \sum_{i=T+1}^{maxValue} h(i)$$

4. Se determina el nuevo valor de umbral T.

$$T = (\mu 1 + \mu 2)/2$$

5. Se deben repetir los pasos 2-4 hasta que el valor de T no varíe.

DILATACIÓN Y EROSIÓN

IMAGEN ORIGINAL

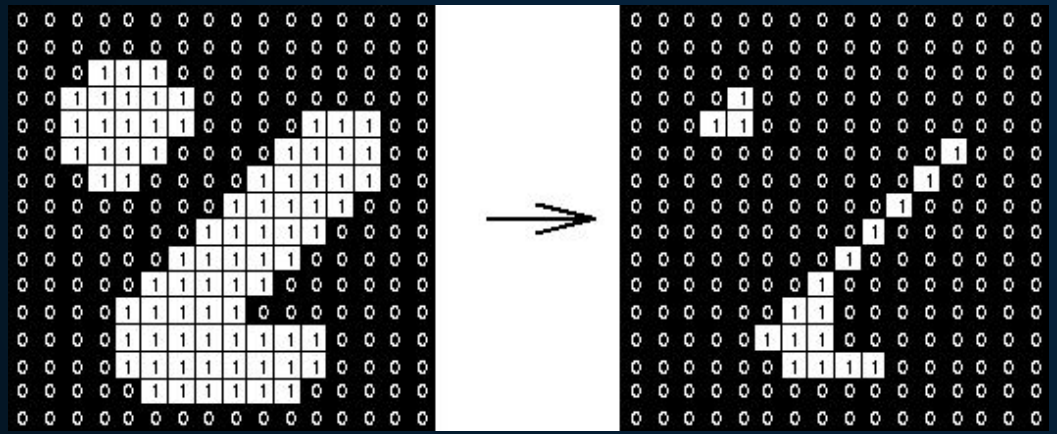
THRESHOLD

DILATACIÓN

EROSIÓN

EROSIÓN

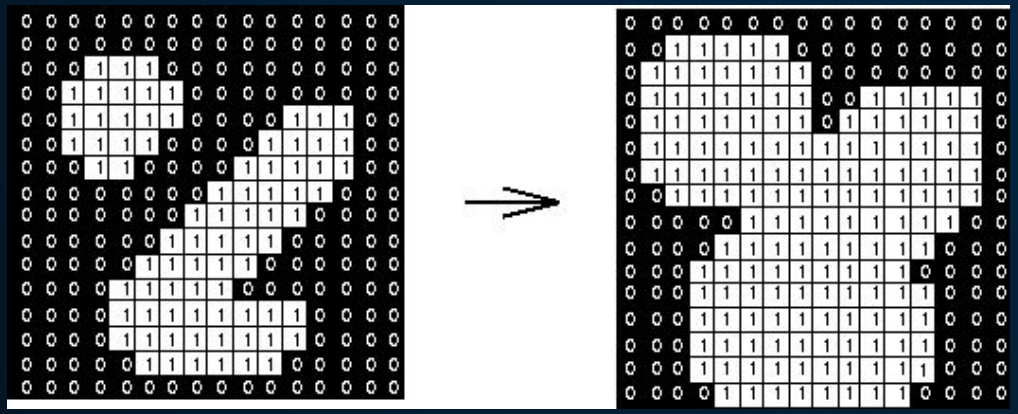
La erosión es una de las dos operaciones morfológicas fundamentales (la otra es la dilatación). La operación de erosión suele utilizar un elemento estructurador para sondear y reducir las formas contenidas en la imagen de entrada.



1	1	1
1	1	1
1	1	1

DILATACIÓN

La dilatación es una de las dos operaciones morfológicas fundamentales (la otra es la erosión). La operación de dilatación suele utilizar un elemento estructurador para sondear y aumentar las formas contenidas en la imagen de entrada.

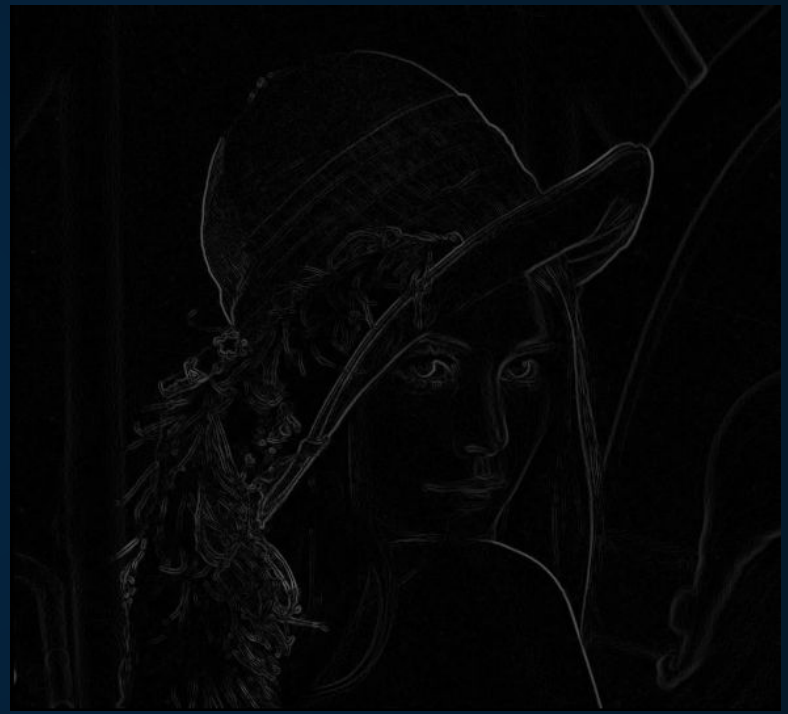


1	1	1
1	1	1
1	1	1

SOBEL

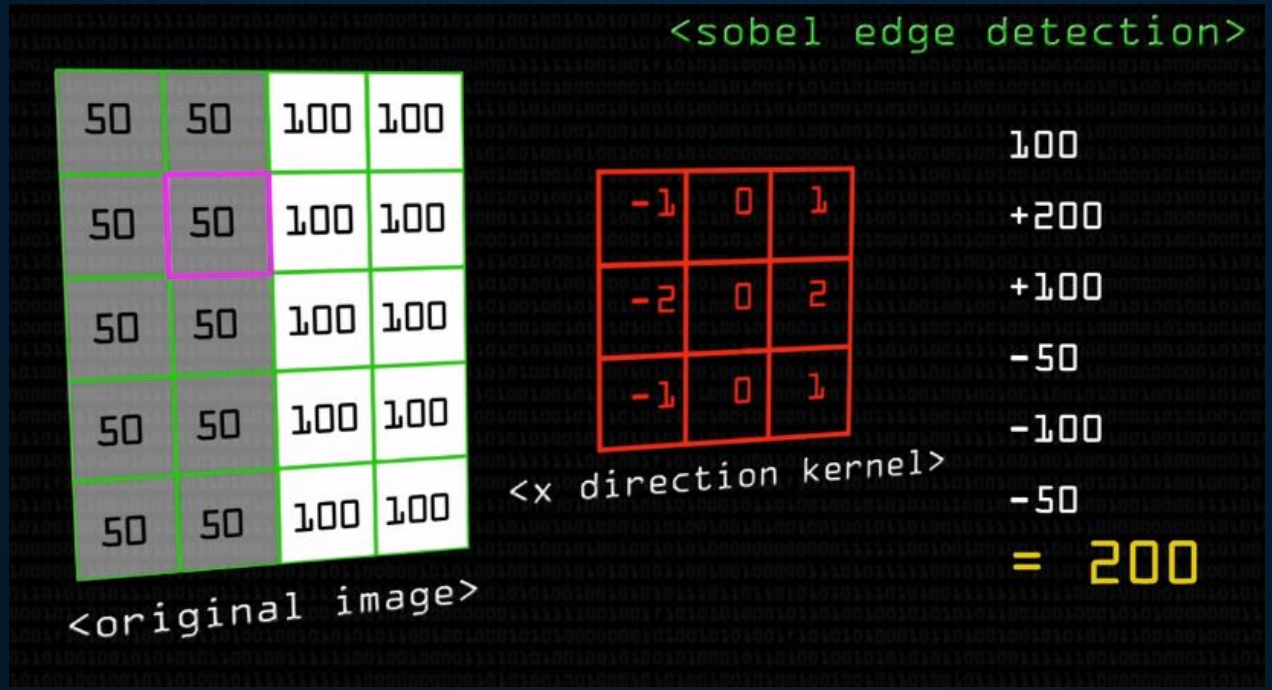
El operador Sobel es utilizado en procesamiento de imágenes, especialmente en algoritmos de detección de bordes. Técnicamente es un operador diferencial discreto que calcula una aproximación al gradiente de la función de intensidad de una imagen. Para cada punto de la imagen a procesar, el resultado del operador Sobel es tanto el vector gradiente correspondiente como la norma de este vector.

El operador Sobel calcula el gradiente de la intensidad de una imagen en cada punto (píxel).



SOBEL

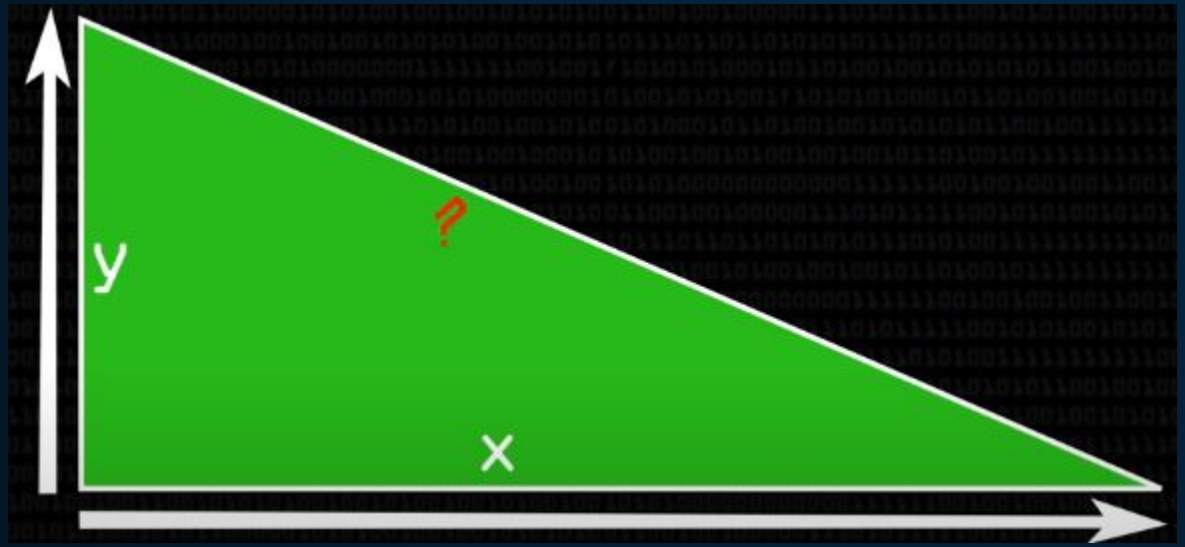
$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ 2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{y} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$



SOBEL

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$



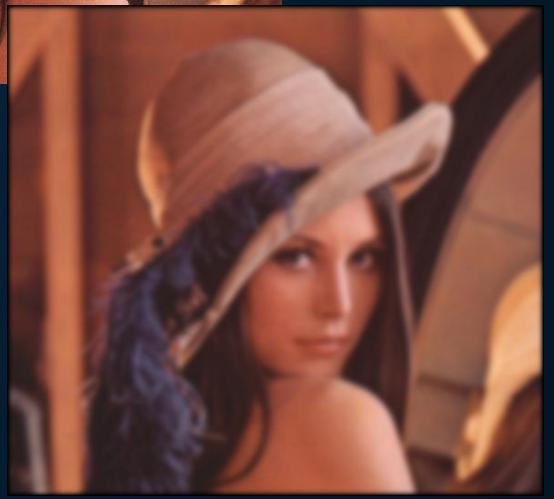
Otros operadores para la detección de bordes :

- Prewitt Operator
- Robinson Compass Masks
- Krisch Compass Masks
- Laplacian Operator

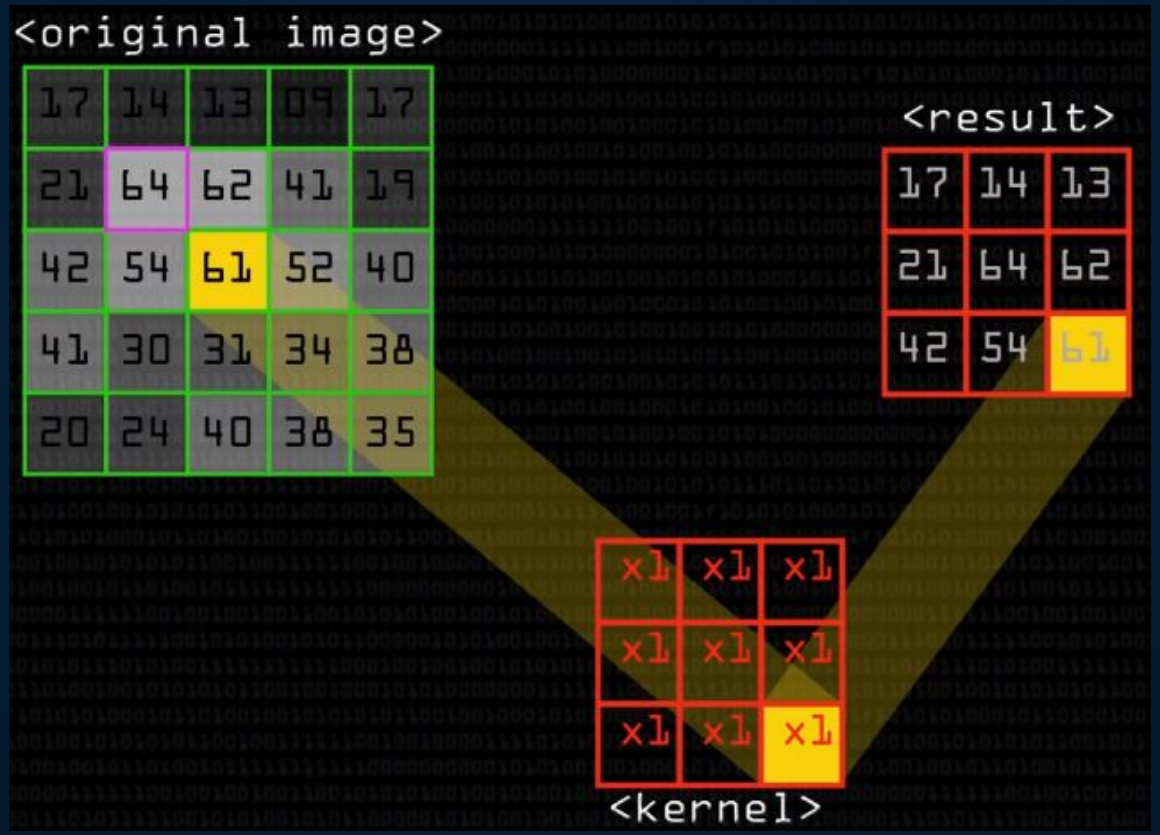
MEAN BLUR

Difumina la imagen, por medio de la convolución de la matrix de la imagen con una máscara (kernel) específico.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



MEAN BLUR

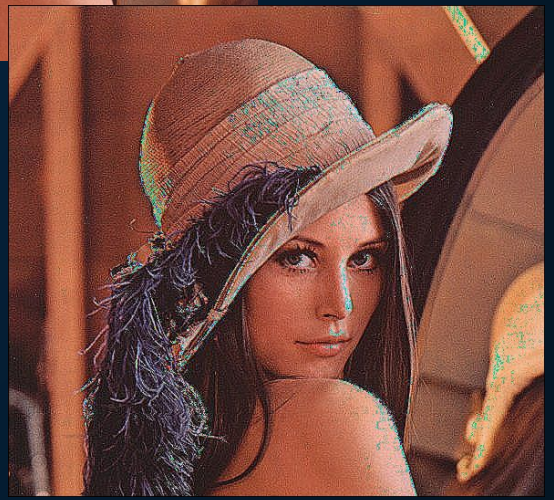

<original image>
17 14 13 09 17
21 64 62 41 19
42 54 61 52 40
41 30 31 34 38
20 24 40 38 35
<result>
17 14 13
21 64 62
42 54 61
x1 x1 x1
x1 x1 x1
x1 x1 x1
<kernel>

MEAN BLUR

SHARPEN

Realza los bordes de la imagen, por medio de la convolución de la matrix de la imagen con una máscara (kernel) específico.

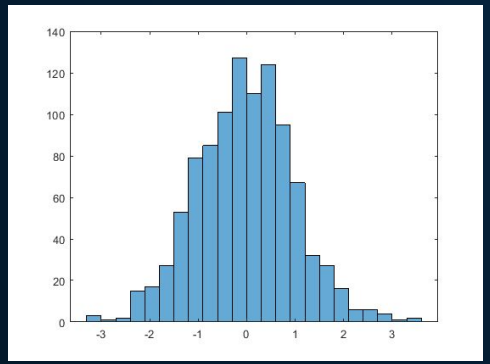
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

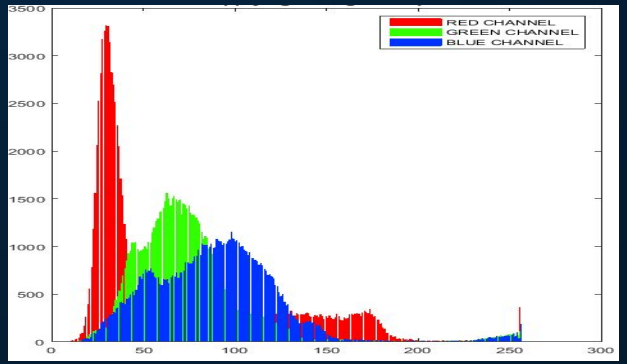


HISTOGRAMA

Un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados; dicha representación muestra la distribución del color en una imagen.

Representa el número de pixeles que poseen cierta intensidad dentro de una lista de rangos de colores, que se extienden sobre el espacio de color de la imagen





HISTOGRAMA

```
ImagePPM i1("../proyecto/images/lenna_PPM.ppm");
```

```
i1.histogramPPM("green");
```

```
0: 7 ; 1: 0 ; 2: 2 ; 3: 2 ; 4: 0 ; 5: 6 ; 6: 7 ; 7: 6 ; 8: 13 ; 9: 12 ; 10: 12 ; 11: 23 ; 12: 37 ; 13: 27 ; 14: 48 ; 15: 68 ; 16: 101 ; 17: 110 ; 18: 148 ; 19: 192 ; 20: 281 ; 21: 33
8 ; 22: 472 ; 23: 671 ; 24: 928 ; 25: 1321 ; 26: 1863 ; 27: 2586 ; 28: 3654 ; 29: 5081 ; 30: 5748 ; 31: 6359 ; 32: 6283 ; 33: 5780 ; 34: 5082 ; 35: 4501 ; 36: 4095 ; 37: 3670 ; 38: 3
392 ; 39: 3053 ; 40: 2942 ; 41: 2680 ; 42: 2497 ; 43: 2398 ; 44: 2481 ; 45: 2383 ; 46: 2524 ; 47: 2599 ; 48: 2783 ; 49: 2924 ; 50: 2964 ; 51: 3072 ; 52: 3224 ; 53: 3163 ; 54: 3030 ;
55: 2842 ; 56: 2746 ; 57: 2557 ; 58: 2426 ; 59: 2252 ; 60: 2186 ; 61: 2119 ; 62: 2018 ; 63: 2094 ; 64: 2083 ; 65: 2187 ; 66: 2118 ; 67: 2148 ; 68: 2196 ; 69: 2199 ; 70: 2254 ; 71: 24
44 ; 72: 2424 ; 73: 2568 ; 74: 2467 ; 75: 2595 ; 76: 2610 ; 77: 2522 ; 78: 2707 ; 79: 2566 ; 80: 2537 ; 81: 2588 ; 82: 2624 ; 83: 2700 ; 84: 2805 ; 85: 2919 ; 86: 2849 ; 87: 2865 ; 8
8: 3020 ; 89: 3115 ; 90: 3195 ; 91: 3164 ; 92: 3134 ; 93: 3023 ; 94: 2995 ; 95: 2908 ; 96: 2903 ; 97: 2915 ; 98: 3078 ; 99: 3308 ; 100: 3358 ; 101: 3523 ; 102: 3518 ; 103: 3335 ; 104
: 3135 ; 105: 2799 ; 106: 2599 ; 107: 2268 ; 108: 2121 ; 109: 1979 ; 110: 1915 ; 111: 1814 ; 112: 1744 ; 113: 1801 ; 114: 1701 ; 115: 1628 ; 116: 1500 ; 117: 1383 ; 118: 1285 ; 119:
1258 ; 120: 1195 ; 121: 1134 ; 122: 1071 ; 123: 895 ; 124: 856 ; 125: 803 ; 126: 698 ; 127: 660 ; 128: 656 ; 129: 672 ; 130: 652 ; 131: 582 ; 132: 589 ; 133: 602 ; 134: 563 ; 135: 58
6 ; 136: 583 ; 137: 590 ; 138: 597 ; 139: 597 ; 140: 628 ; 141: 593 ; 142: 648 ; 143: 658 ; 144: 632 ; 145: 682 ; 146: 755 ; 147: 762 ; 148: 743 ; 149: 811 ; 150: 750 ; 151: 803 ; 15
2: 787 ; 153: 754 ; 154: 740 ; 155: 762 ; 156: 687 ; 157: 657 ; 158: 690 ; 159: 700 ; 160: 701 ; 161: 691 ; 162: 644 ; 163: 710 ; 164: 722 ; 165: 722 ; 166: 667 ; 167: 765 ; 168: 683
 ; 169: 702 ; 170: 785 ; 171: 736 ; 172: 703 ; 173: 751 ; 174: 778 ; 175: 713 ; 176: 713 ; 177: 697 ; 178: 656 ; 179: 618 ; 180: 494 ; 181: 465 ; 182: 403 ; 183: 354 ; 184: 323 ; 185
: 265 ; 186: 228 ; 187: 224 ; 188: 172 ; 189: 153 ; 190: 155 ; 191: 146 ; 192: 115 ; 193: 112 ; 194: 90 ; 195: 76 ; 196: 74 ; 197: 65 ; 198: 47 ; 199: 38 ; 200: 38 ; 201: 24 ; 202: 3
7 ; 203: 18 ; 204: 21 ; 205: 13 ; 206: 11 ; 207: 7 ; 208: 9 ; 209: 6 ; 210: 5 ; 211: 2 ; 212: 3 ; 213: 3 ; 214: 2 ; 215: 2 ; 216: 3 ; 217: 1 ; 218: 1 ; 219: 1 ; 220: 0 ; 221: 0 ; 222
: 0 ; 223: 0 ; 224: 2 ; 225: 1 ; 226: 0 ; 227: 0 ; 228: 0 ; 229: 2 ; 230: 0 ; 231: 0 ; 232: 0 ; 233: 0 ; 234: 0 ; 235: 0 ; 236: 0 ; 237: 0 ; 238: 0 ; 239: 0 ; 240: 0 ; 241: 0 ; 242:
0 ; 243: 0 ; 244: 0 ; 245: 0 ; 246: 0 ; 247: 0 ; 248: 0 ; 249: 0 ; 250: 0 ; 251: 0 ; 252: 0 ; 253: 0 ; 254: 0 ; 255: 0 ;
```

COMBINACIÓN

Combinación busca juntar dos imágenes en una sola haciendo uso del chromakey, para así obtener una sola imagen resultante con las dos en ella, esto suele usarse mucho en la edición de imágenes, similar al tener una imagen PNG con fondo transparente.

```
void ImagePPM::chromaKey(ImagePPM &other)
```

```
int chromaRed = round( ( other.red[0] + other.red[1] + other.red[other.cols] + other.red
[other.cols - 1] + other.red[other.cols - 2] + other.red[(2*other.cols) - 1] + other.red[
(other.rows - 1)*other.cols] + other.red[(other.rows - 1)*other.cols + 1] ) / 8.0 );

int chromaGreen = round( ( other.green[0] + other.green[1] + other.green[other.cols] + other.
green[other.cols - 1] + other.green[other.cols - 2] + other.green[(2*other.cols) - 1] + other.
green[(other.rows - 1)*other.cols] + other.green[(other.rows - 1)*other.cols + 1] ) / 8.0 );

int chromaBlue = round( ( other.blue[0] + other.blue[1] + other.blue[other.cols] + other.blue
[other.cols - 1] + other.blue[other.cols - 2] + other.blue[(2*other.cols) - 1] + other.blue[
(other.rows - 1)*other.cols] + other.blue[(other.rows - 1)*other.cols + 1] ) / 8.0 );


int foregroundBegin_row = floor( (( 1 -  other.rows/double(rows))/2.0) * rows ) ;
int foregroundBegin_col = floor( (( 1 -  other.cols/double(cols))/2.0) * cols ) ;
```

Redondea el componente de los canales y hace el promedio para obtener el valor del color del fondo (verde).

Ubica la imagen (pasada por ref) de forma centrada en la imagen de fondo.

Busca el valor candidato de fondo verde y así reemplazarlo por una mayor pureza, para luego poder ser reconocido eficazmente.

```
for(int i=0; i<other.red.size(); i++){

  if( (chromaGreen - 10) <= int(other.green[i])  &&  int(other.green[i]) <= (chromaGreen +
  10) ){

    if( (chromaBlue - 10) <= int(other.blue[i])  &&  int(other.blue[i]) <= (chromaBlue + 10) )
    {

      if( (chromaRed - 10) <= int(other.red[i])  &&  int(other.red[i]) <= (chromaRed + 10) ){
        other.red[i] = 0;
        other.green[i] = 255;
        other.blue[i] = 0;
      }
    }
  }

}
```

```
std::vector<unsigned int> tempRed ( red.size() );
std::vector<unsigned int> tempGreen ( green.size() );
std::vector<unsigned int> tempBlue ( blue.size() );

tempRed = red;
tempGreen = green;
tempBlue = blue;

int contador=0;
for(int i=foregroundBegin_row; i < (foregroundBegin_row + other.rows); i++){
  for(int j=foregroundBegin_col; j < (foregroundBegin_col + other.cols); j++){

    red[(cols*i) + j] = other.red[contador];
    green[(cols*i) + j] = other.green[contador];
    blue[(cols*i) + j] = other.blue[contador];

    contador++;

    }
  }
```

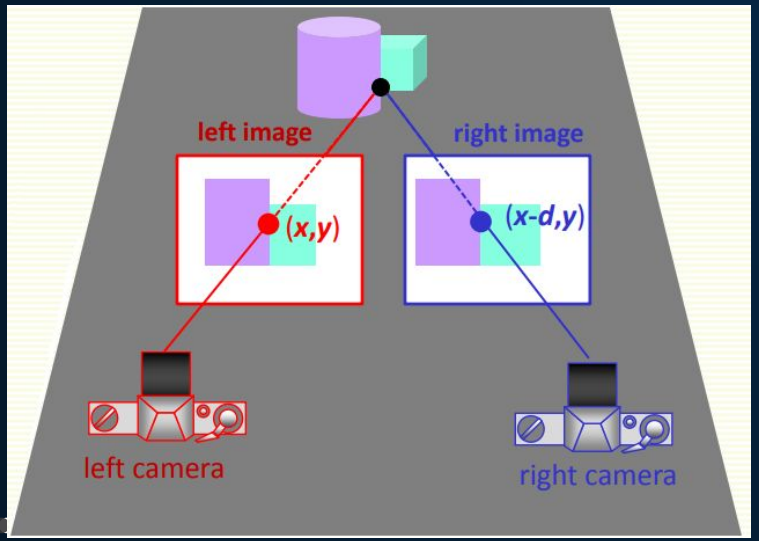
Se asignan vectores temporales, los cuales tomarán los valores de la imagen de fondo (plazoleta ingeniería). El ciclo recorre la imagen (ardilla) y la posiciona en la imagen de fondo.

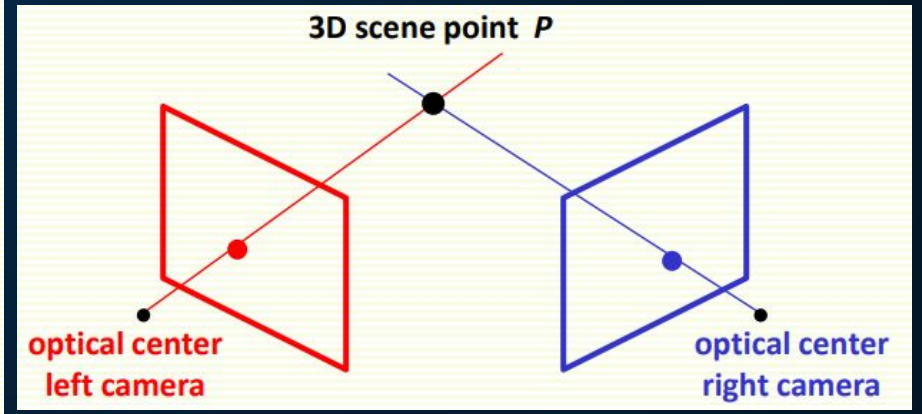
Busca en la imágen completa los valores de pureza del verde, para así reemplazarlo con los valores de la imagen del fondo.

```
for(int i=0; i<red.size(); i++){

  if( int(green[i]) == 255 ){

    if( int(blue[i]) == 0 ){

      if( int(red[i]) == 0 ){

        red[i] = tempRed[i];
        green[i] = tempGreen[i];
        blue[i] = tempBlue[i];
      }
    }
  }

}
```

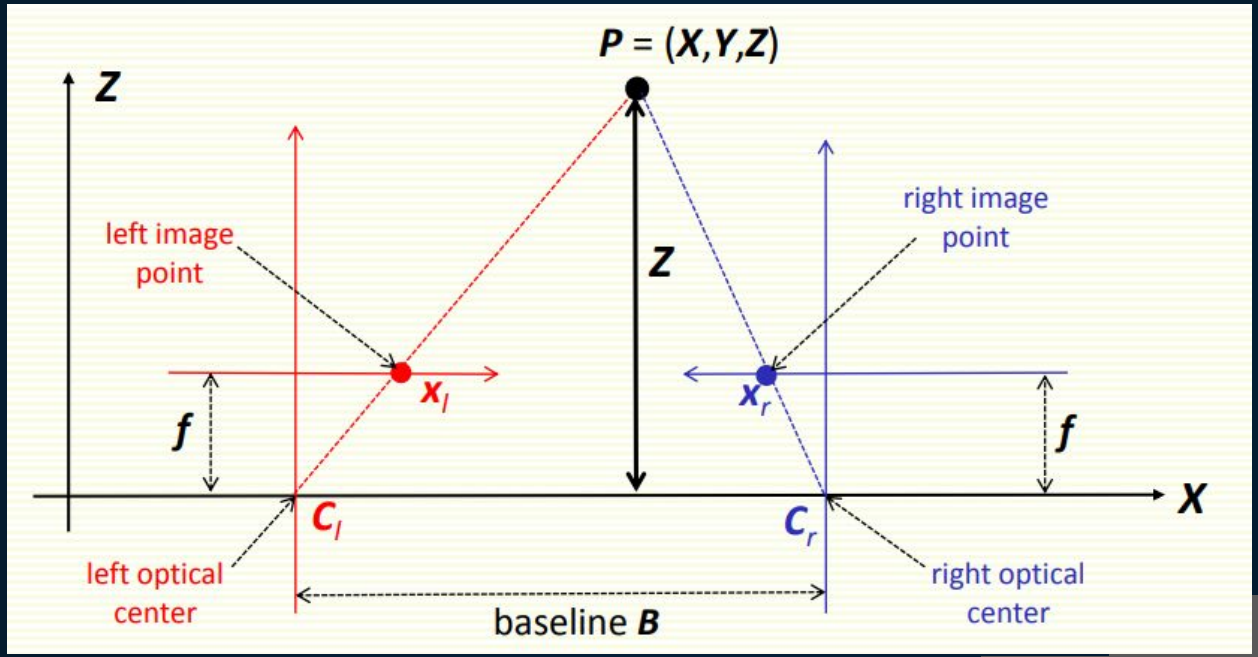
DISPARIDAD BINOCULAR

Se conoce como disparidad binocular a la ligera diferencia entre los dos puntos de vista proporcionados por dos cámaras. La disparidad binocular es la forma de percibir profundidad y/ó relieve.





Para la implementación del algoritmo partimos del hecho de que las dos imágenes estéreo ingresadas al programa ya estaban rectificadas.



Para determinar la correspondencia entre los píxeles de ambas imágenes se utilizó el método SAD (sum of absolute differences)

- Sum of absolute differences: $\sum\sum |L(r,c) - R(r,c-d)|$

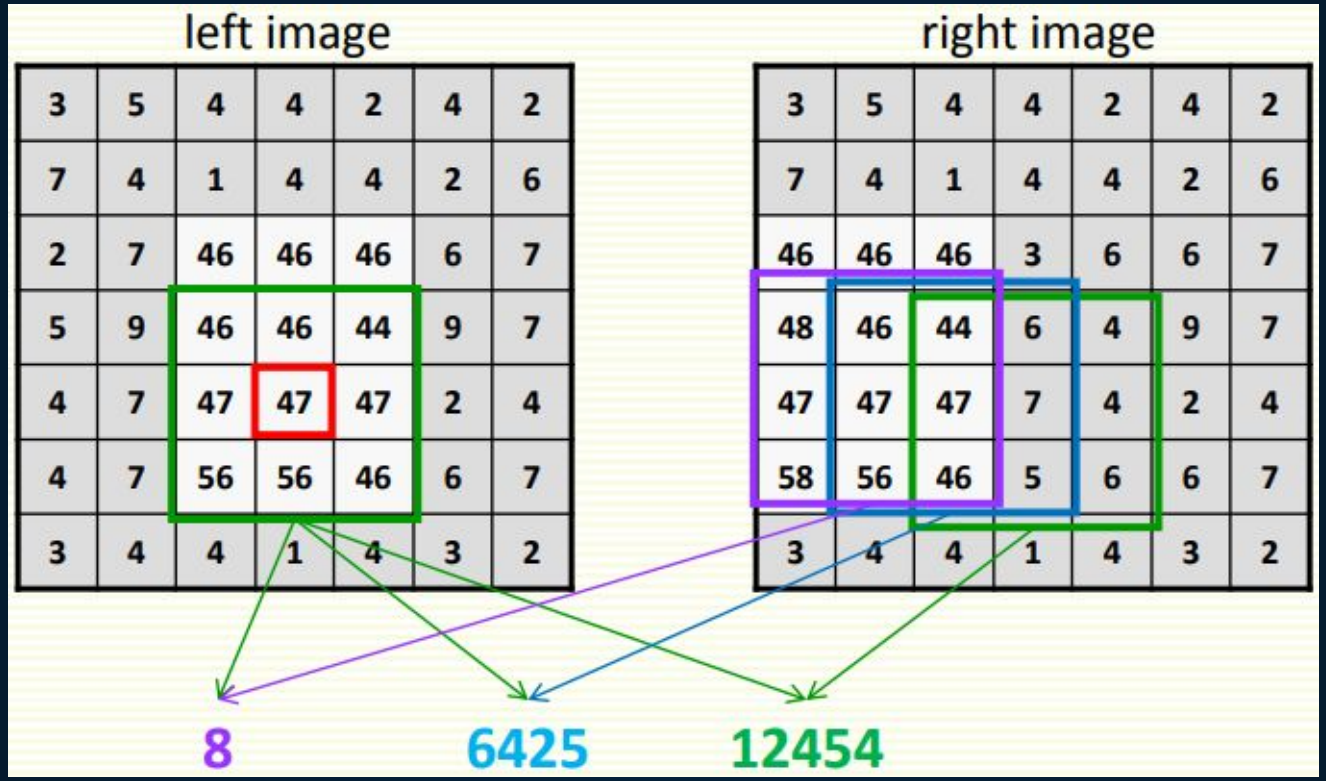
Template	Search image
2 5 5	2 7 5 8 6
4 0 7	1 7 4 2 7
7 5 9	8 4 6 8 5

Left	Center	Right
0 2 0	5 0 3	3 3 1
3 7 3	3 4 5	0 2 0
1 1 3	3 1 1	1 3 4

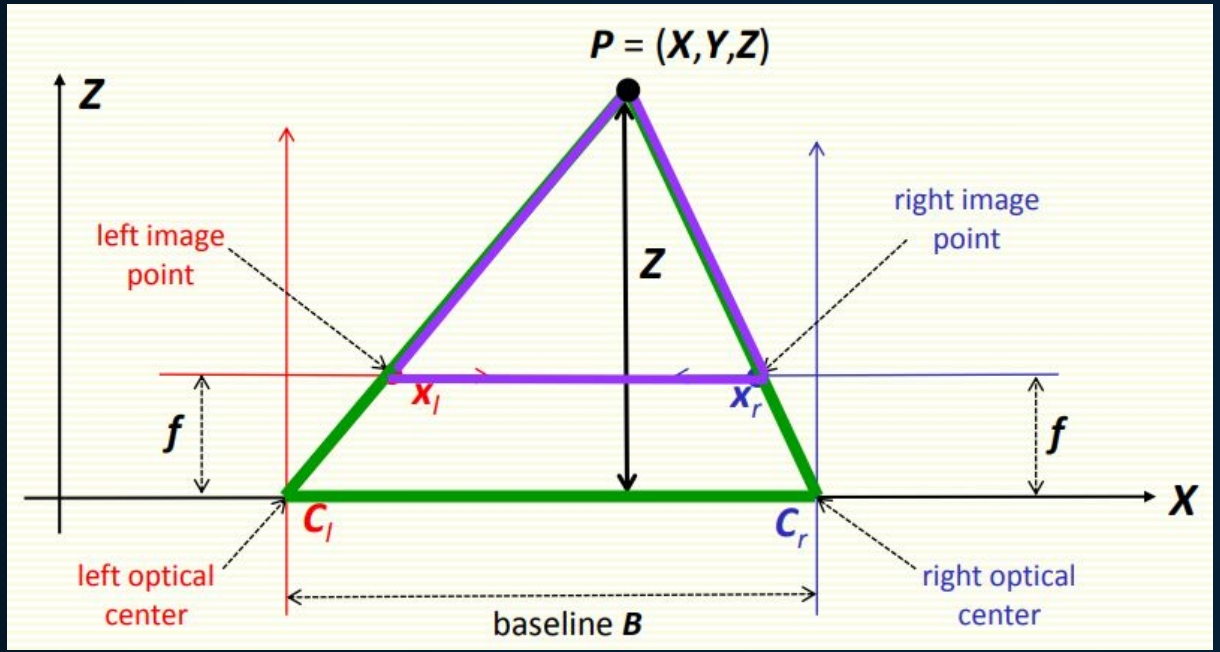
left = 20

center = 25

right = 17


left image
right image
3 5 4 4 2 4 2
7 4 1 4 4 2 6
2 7 46 46 46 6 7
5 9 46 46 44 9 7
4 7 47 47 47 2 4
4 7 56 56 46 6 7
3 4 4 1 4 3 2
3 5 4 4 2 4 2
7 4 1 4 4 2 6
46 46 46 3 6 6 7
48 46 44 6 4 9 7
47 47 47 7 4 2 4
58 56 46 5 6 6 7
3 4 4 1 4 3 2
8
6425
12454

Aplicando semejanza de triángulos:



$$\frac{Z}{B} = \frac{Z - f}{B - x_l + x_r}$$

$$Z = \frac{B \cdot f}{x_l - x_r}$$

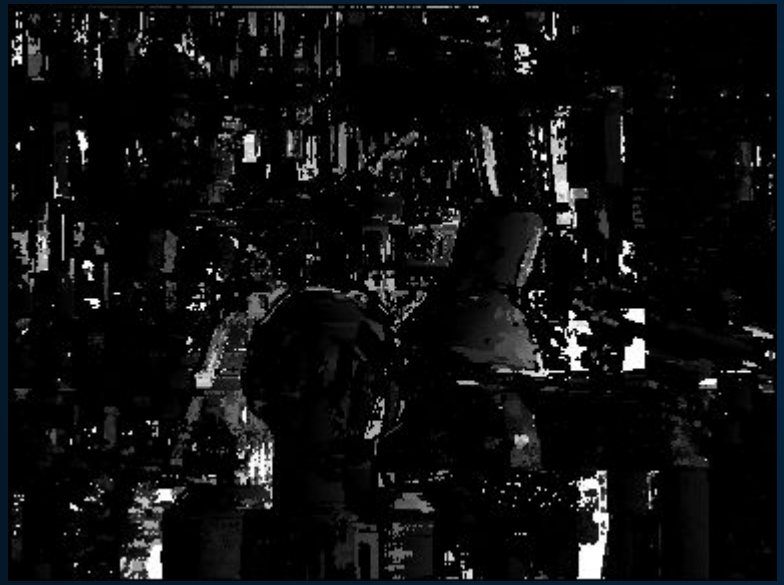
Aplicación y resultados:

Left Image

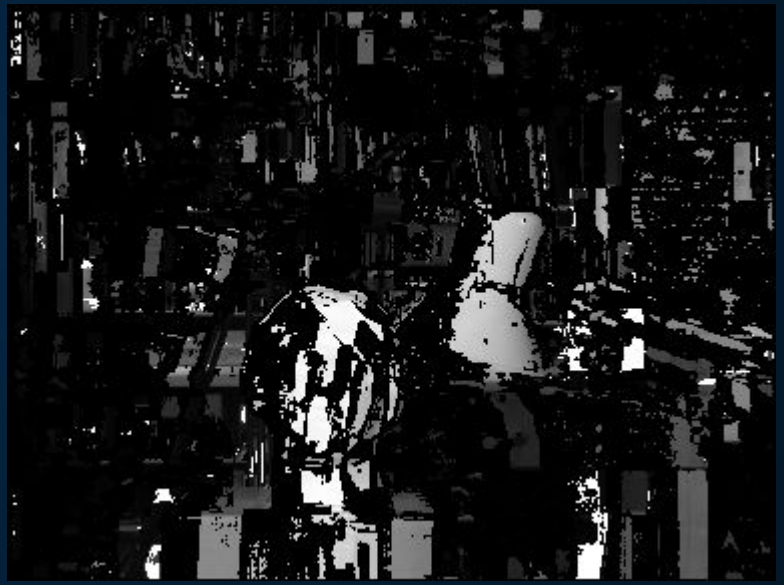
Right Image

Aplicación y resultados:

Disparity Image

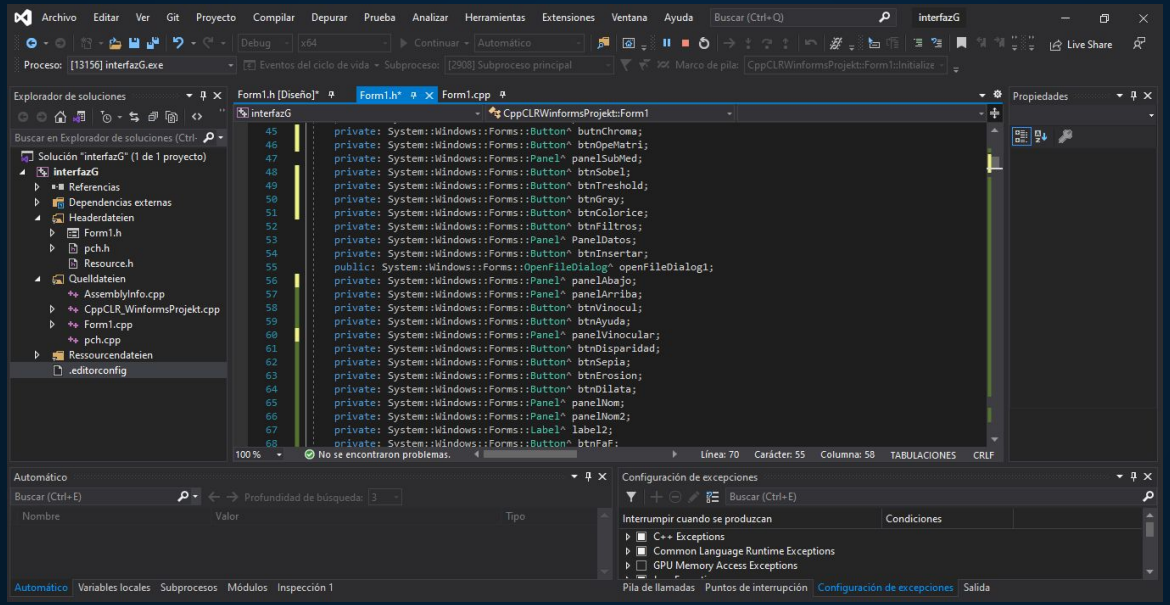


Depth perception Image

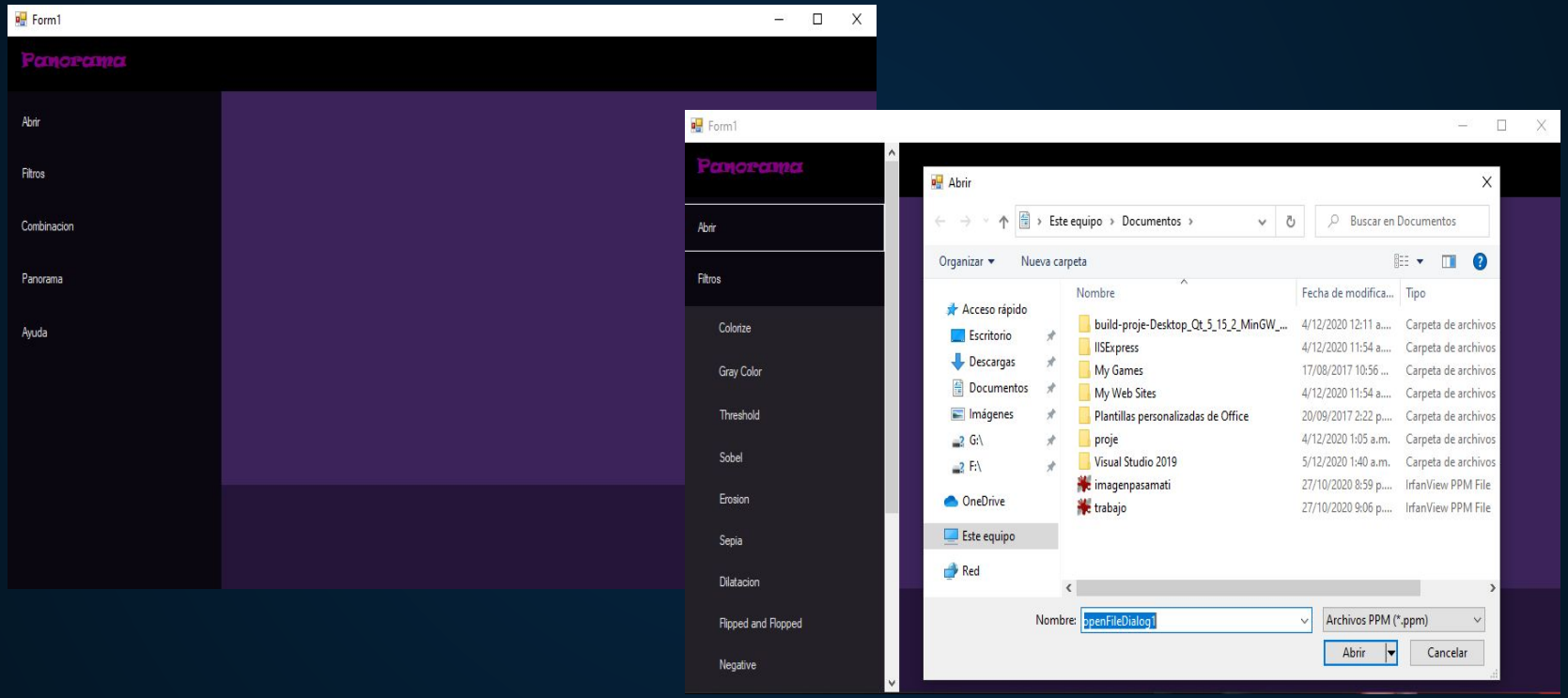


Prototipo de interfaz gráfica

Para la interfaz gráfica utilizamos la herramienta Windows Forms de Microsoft, la cual nos permitió tener un diseño extensible, flexible y moderno.



Extensible, flexible y moderna.



HERRAMIENTAS PARA EL DESARROLLO:

C++

Se usó el lenguaje de programación C++ con paradigma orientado a objetos para realizar el proyecto.

01

VISUAL STUDIO CODE

El programa se realizó en el ide de visual studio codes ya que este nos permite trabajar con el código de una manera más eficaz.

02

EXTENSIÓN GIT DUCK

Esta extensión nos permite realizar la programación de nuestro proyecto de manera sincrónica y cooperativa.

03

04

DISCORD

Este programa nos permitió tener una buena comunicación entre el equipo.

05

GIT HUB

Este nos ayudó como repositorio y planificación del proyecto.

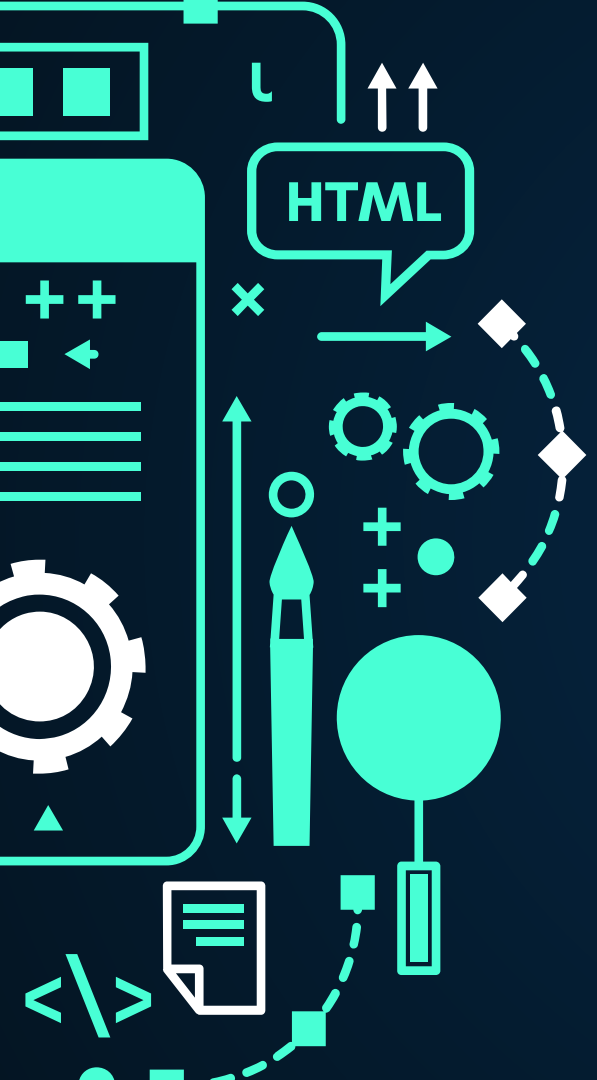
06

GOOGLE DRIVE

Estas herramientas permite tener acceso a la información necesaria alojada en la nube.

WEBGRAFIA

- https://www.uv.es/gpoei/eng/Pfc_web/generalidades/pseudocolor/pseudocolor.htm
- https://allielearning.wordpress.com/2017/10/21/pseudocolor-image-processing/
- http://webdiis.unizar.es/~neira/12082/thesholding.pdf
- https://eprints.ucm.es/18030/1/T34207.pdf
- https://www.youtube.com/watch?v=uihBwtPIBxM
- https://es.wikipedia.org/wiki/Histograma
- https://es.wikipedia.org/wiki/Histograma_de_color
- https://www.youtube.com/watch?v=C_zFhWdM4ic&t=171s
- https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm
- https://towardsdatascience.com/depth-estimation-1-basics-and-intuition-86f2c9538cd1
- https://www.csd.uwo.ca/~oveksler/Courses//Winter2016/CS4442_9542b/L11-CV-stereo.pdf
- http://bibing.us.es/proyectos/abreproy/70265/fichero/Cap%C3%ADtulo+5.pdf
- http://www-scf.usc.edu/~anishver/assignments/cs103/PA2%20-%20Chroma%20Key.pdf
- https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e#:~:text=Canny%20edge%20detector%20is%20probably,Gaussian%20filter%20to%20reduce%20noise.
- http://netpbm.sourceforge.net/doc/ppm.html
- https://www.rapidtables.com/convert/image/rgb-to-grayscale.html
- https://www.tutorialspoint.com/dip/sobel_operator.htm



¡GRACIAS!