



Rotación de imágenes digitales.

Tratamiento Digital de Imágenes.

24 MAYO

Universidad de Almería

Creado por: Francisco José López Carrillo



UNIVERSIDAD
DE ALMERÍA



Rotación de Imágenes digitales.

Índice.

| | | |
|-------|--|----|
| 1. | ¿Qué es la rotación de una imagen?..... | 5 |
| 2. | Preparación de las fotos. | 5 |
| 3. | Rotaciones básicas. | 6 |
| 4. | Rotación Θ | 7 |
| 5. | Problemas en la rotación..... | 8 |
| 6. | Cálculo de la nueva imagen. | 11 |
| 7. | Traslado al código..... | 14 |
| 7.1. | Rotación de 0° a 90° | 14 |
| 7.2. | Rotación de 91° a 180° | 17 |
| 7.3. | Rotación 181° a 270° | 19 |
| 7.4. | Rotación 271° a 360° | 21 |
| 8. | Otras rotaciones..... | 22 |
| 8.1. | Rotación espejo Vertical. | 23 |
| 8.2. | Rotación espejo Horizontal. | 24 |
| 9. | Anti – Aliasing..... | 24 |
| 9.1. | Filtro de la media..... | 26 |
| 9.2. | Filtro de la mediana. | 28 |
| 9.3. | Filtro de Gauss..... | 29 |
| 9.4. | Three Shears. | 30 |
| 9.5 | Solución Aliasing..... | 31 |
| 10. | Ejemplo de ejecución en C++ | 32 |
| 10.1. | Ejecución común..... | 33 |
| 10.2. | Opción Rotación. | 33 |
| 10.3. | Opción espejo. | 34 |
| 11. | Conclusión. | 35 |
| 12. | Bibliografía..... | 35 |



1. ¿Qué es la rotación de una imagen?

Si buscamos la definición exacta de rotación, nos derivamos a la RAE que nos dice que la rotación es “*dar vueltas alrededor de un eje*”, por tanto, las rotaciones de imágenes son exactamente esto, elegimos un eje en el que rotar y la rotamos tantos grados como deseamos.

Una imagen es una matriz donde en cada coordenada nos encontramos con unos valores. En este proyecto todas las imágenes las trataremos como matrices y haremos las operaciones de matrices para llegar al resultado deseado.

2. Preparación de las fotos.

Las fotos que utilizamos en esta práctica son fotos en formato .bmp (*bit map*), como su nombre indica, es un mapa de bits, pertenece a Windows y no tiene compresión, este sistema puede guardar imágenes de 24 bits (millones de colores), 16 bits, 8 bits o menos, en este caso nos interesa el de 8 bits para obtener nuestra imagen en 256 colores en escala de grises.

Para crear las imágenes BMP vamos a usar un convertidor online, podemos introducir cualquier imagen .JPG (imagen con compresión) y eligiendo 8 bits, la tendremos lista para nuestro proyecto.

El convertidor es <https://online-converting.com/image/convert2bmp/>

The screenshot shows the 'BMP converter' website. At the top, there is a large box with a blue button labeled 'Add image files' (circled in red with a '2'), the text 'Click here to choose files or Drop them here', and 'Up to 50 files'. Below this is a table with columns '#', 'Source file', 'Size', and 'Status'. The first row shows a file named 'SirPerro.jpeg' with a size of '49.9 KB'. In the 'Status' column, there is a blue link 'Download SirPerro.bmp' (circled in red with a '3') and two other links 'Info' and 'Delete from server'. Below the table, there are three dropdown menus: 'Color:' set to '8 (Indexed)' (circled in red with a '1'), 'With rows direction:' set to 'Bottom - Top (default)', and 'Quantization equal to:' set to '8'.

| # | Source file | Size | Status |
|---|---------------|---------|---|
| 1 | SirPerro.jpeg | 49.9 KB | Download SirPerro.bmp Info Delete from server |

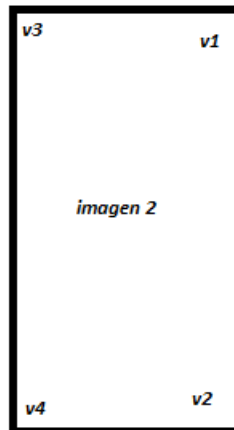
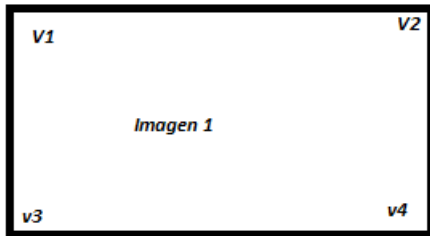
Color: 8 (Indexed) 1
With rows direction: Bottom - Top (default)
Quantization equal to: 8

3. Rotaciones básicas.

Las rotaciones más básicas son 90° y 180° . Para calcular la operación de rotación de 90 grados solo necesitamos que la altura la imagen1 sea igual a $M \times N$, y que la imagen 2 sea $N \times M$.

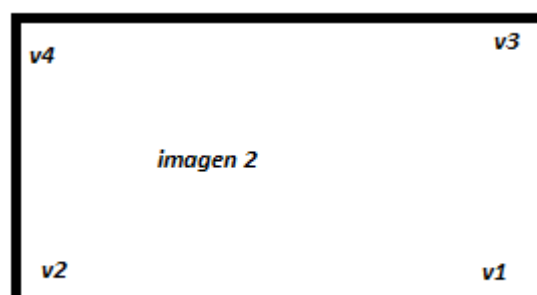
$M \Rightarrow$ Son las filas totales de la imagen.

$N \Rightarrow$ Son las columnas totales de la imagen.



```
for(i = 1; i <= N; i++)
  for(j = 1; j <= M; j++)
    Imagen2(i,j) = imagen1(j,N-i+1)
```

La rotación en 180° es muy parecida a la de 90° pero la imagen 1 es $M \times N$ y la imagen 2 es también $M \times N$. Por tanto, se queda del siguiente modo:



```
for(i = 1; i <= N; i++)  
    for(j = 1; j <= M; j++)  
        Imagen2(i,j) = imagen1(M-i+1,N-j+1)
```

Pero es claro, 90 y 180 grados no abarca toda la gama de grados que tiene una circunferencia, es por ello por lo que debemos usar una fórmula que abarque todos los grados.

4. Rotación Θ

Para poder realizar las operaciones de rotación usamos la matriz de rotación, que es la siguiente:

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}.$$

Esta fórmula representa la rotación en un espacio euclídeo, esta matriz representa las rotaciones antihorarias en muchos campos.

Si queremos aplicar esta fórmula en unas coordenadas entonces tenemos que aplicar una multiplicación por x e y para obtener a x' e y'.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

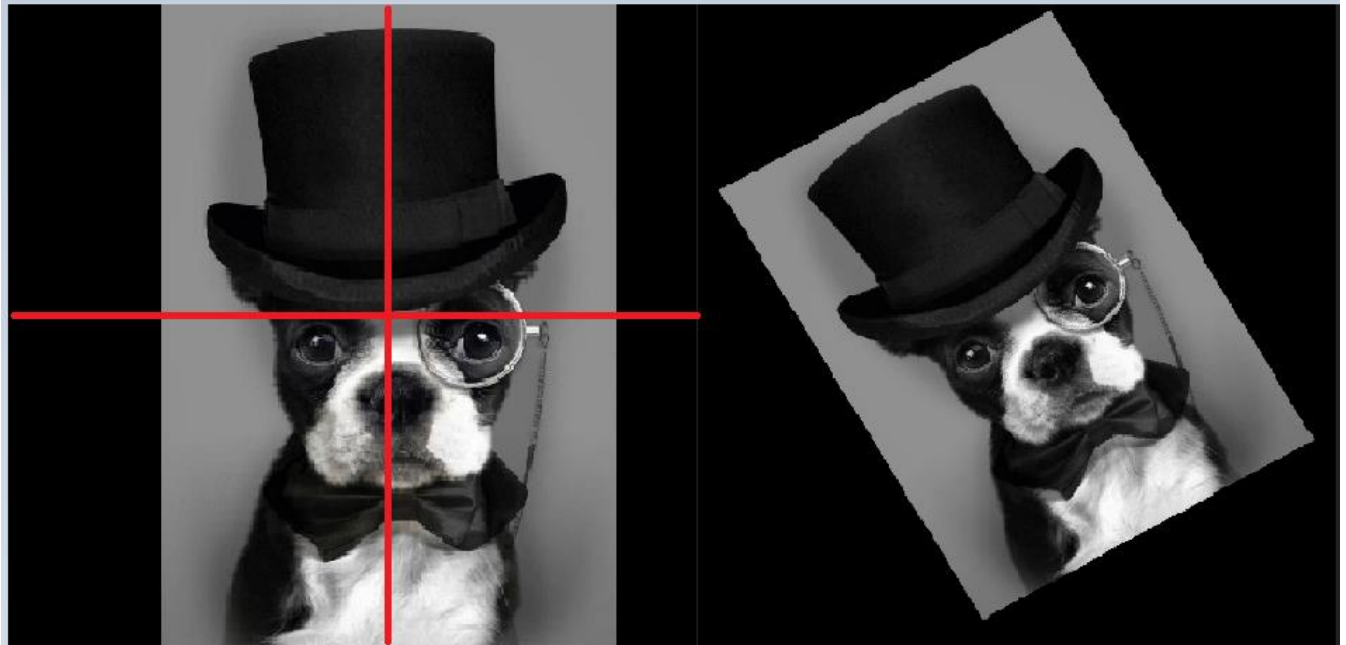
Quedando la fórmula de coordenadas de la siguiente manera:

$$\begin{aligned} x' &= x \cos \theta + y \sin \theta, \\ y' &= -x \sin \theta + y \cos \theta. \end{aligned}$$

5. Problemas en la rotación.

Entonces actualmente ya tenemos la “solución” a nuestro problema de la rotación, teniendo una fórmula que nos soluciona las coordenadas a las que debemos estar.

Pero no es así, el comportamiento de esta matriz está creado para cuando tenemos matrices con su punto (0,0) en el centro de la imagen, como se muestra en el siguiente ejemplo:



La imagen rotada se encuentra rotada a 30°, en sentido antihorario.

Por tanto, esta fórmula es muy útil para cuando nuestro centro es la coordenada (0,0), pero cuando no es, la imagen gira en su otro eje, por ejemplo, en el extremo arriba izquierda, comenzando por (1,1).

En nuestro programa se puede especificar cuál es el centro de la imagen, pero para este proyecto he elegido hacerlo a partir del punto (1,1).

Imagen normal:

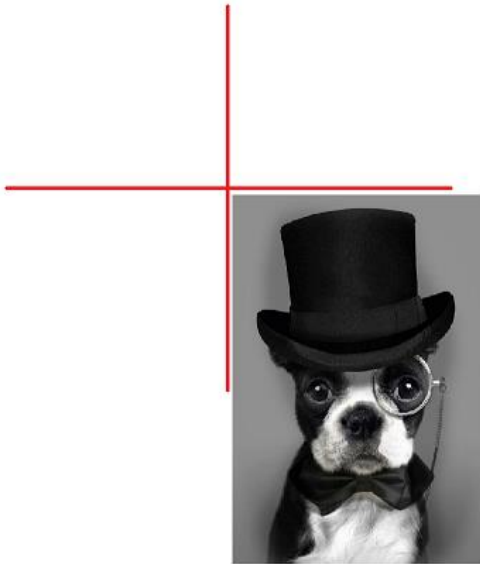


Imagen girada 90°:

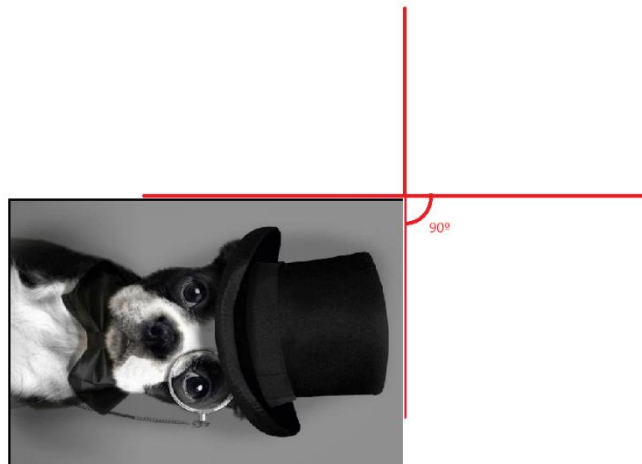


Imagen girada 180°:

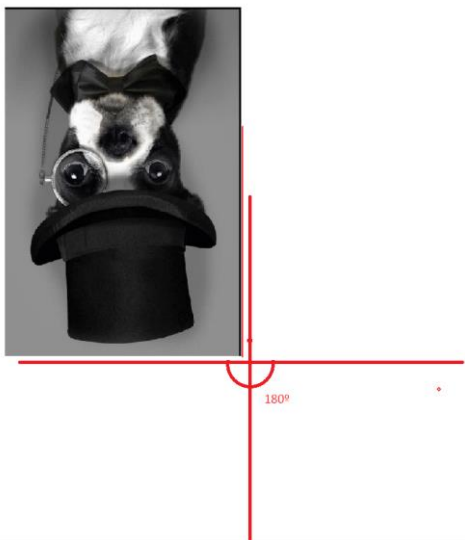
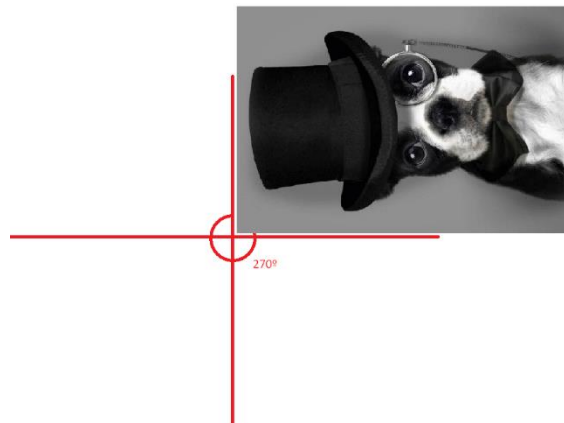


Imagen girada 270°:



Por tanto, nos encontramos que cuando rotamos la imagen se sale del sitio, esto nos induce a un error cuando intentamos programar nuestro algoritmo.

Si no hiciéramos ninguna modificación al algoritmo de rotación, la imagen quedaría así una vez realizada la rotación:



Imagen rotada 40° antihorario.

Este problema lo debemos solucionar para nuestras rotaciones, es por eso por lo que debemos asociar el centro de la imagen para que se rote todo en raíz a este centro calculado, por lo tanto, debemos programar un desplazamiento de la imagen.

Este desplazamiento viene dado por la siguiente fórmula:

$C_x \Rightarrow$ es el centro de imagen2 calculado con $M/2$.

$C_y \Rightarrow$ es el centro de imagen2 calculado con $N/2$.

$$\text{Imagen2}(x, y) = \begin{vmatrix} \cos \Theta & \sin \Theta & c_x - c_x * \cos \Theta - c_y * \sin \Theta \\ -\sin \Theta & \cos \Theta & c_y + c_x * \sin \Theta - c_y * \cos \Theta \\ X \\ Y \\ 1 \end{vmatrix}$$

Esta matriz nos da la siguiente fórmula:

$$X' = x * \cos \Theta + y * \sin \Theta + c_x - c_x * \cos \Theta - c_y * \sin \Theta$$

$$Y' = -x * \sin \Theta + y * \cos \Theta + c_y + c_x * \sin \Theta - c_y * \cos \Theta$$

Gracias a esta fórmula arreglamos muchos de los errores que teníamos anteriormente en la rotación, pero no está finalizado nuestro trabajo.

Tras el uso de este algoritmo, nuestra imagen queda de la siguiente manera:



Imagen rotada 55° horario.

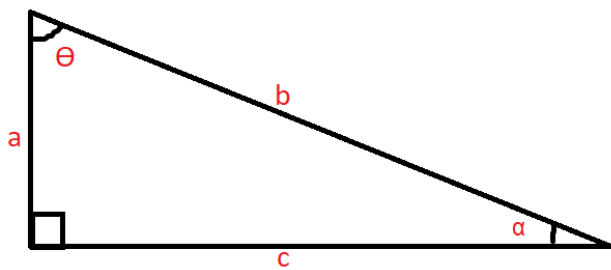
Ahora es cuando tenemos dos opciones en nuestro trabajo, podemos dejarlo así, o podemos hacer que quepa toda la imagen, aunque se muestren partes con píxeles en negro, nos quedamos con la segunda opción.

Para ello vamos a realizar una fórmula que va a calcular del tamaño que tiene que ser nuestra nueva imagen y que no haya ninguna parte que no entre en ella.

6. Cálculo de la nueva imagen.

Primero debemos saber el ángulo que va a tener nuestra nueva imagen, para calcular el tamaño tenemos que hacer cálculos trigonométricos.

Partimos de la base del triángulo rectángulo.



Las fórmulas para sacar todos sus ángulos y lados son:

$$\text{sen } \alpha = a/b$$

$$\text{sen } \Theta = c/b$$

$$\text{cos } \alpha = c/b$$

$$\text{cos } \Theta = a/b$$

$$\text{tan } \alpha = a/c$$

$$\text{tan } \Theta = c/a$$



Imagen rotada 60° antihorario.

El ángulo que conocemos en este caso es $\Theta = 60$ grados. Por tanto, para conocer el nuevo ancho realizaremos la siguiente operación:

$$e = \text{rowN} * \text{sen } \Theta;$$

$$f = \text{ColN} * \text{cos } \Theta;$$

$$\text{Nuevo_Ancho} = e + f = \text{rowN} * \text{sen } \Theta + \text{ColN} * \text{cos } \Theta;$$

Y el nuevo alto:

$$\mathbf{g} = \text{ColN} * \sin \Theta; \quad \mathbf{d} = \text{RowN} * \cos \Theta;$$

$$\mathbf{Nuevo_Alto} = \mathbf{g} + \mathbf{f} = \text{ColN} * \sin \Theta + \text{RowN} * \cos \Theta;$$

Una vez creada esta fórmula nos damos cuenta de que si insertamos ángulos mayores de 90° nos salen mal las medidas, es por ello por lo que para sacar estos datos vamos a normalizar el ángulo para si pasa de 90° calcule como si hubiese vuelto a 0° mediante la función *fmod(angulo,90)* que nos ofrece la librería *cmath* de C++, a parte, vamos a controlar que si el ángulo pasa de 91 hasta los 180° o de 271 a 360° nuestro tamaño se invierte, y pasa a valer Nuevo_Alto = Nuevo_Ancho y Nuevo_Ancho = Nuevo_Alto. Todo esto está en las siguientes líneas de código:

```
//Calculamos el modulo de nuestro angulo, y en caso de ser negativo, lo hacemos positivo
double anguloR = fmod(angulo, 90);
if (anguloR < 0.0) {
    anguloR += 90;
}
//Si tiene valores estos valores el módulo es 0, por ello lo dejamos en 90 para corregir
if (angulo == 90 || angulo == 180 || angulo == 270 || angulo == 360) {
    anguloR = 90;
}
double convertNuevaImagen = (anguloR * (M_PI / 2) / 90);
double sinNewImage = sin(convertNuevaImagen);
double cosNewImage = cos(convertNuevaImagen);
//Invertimos el resultado o no dependiendo de su rango de ángulos.
if (angulo >= 0 && angulo <= 90 || angulo > 180 && angulo <= 270) {
    nAlto = ((sinNewImage * imagen1.LastRow()) + (cosNewImage * imagen1.LastCol()));
    nAncho = ((cosNewImage * imagen1.LastRow()) + (sinNewImage * imagen1.LastCol()));
}
else {
    nAncho = ((sinNewImage * imagen1.LastRow()) + (cosNewImage * imagen1.LastCol()));
    nAlto = ((cosNewImage * imagen1.LastRow()) + (sinNewImage * imagen1.LastCol()));
}
```

Estas líneas solucionan todos los problemas que tienen que ver con el tamaño de la nueva imagen.

7. Traslado al código.

Trasladar toda esta teoría a nuestro código ha sido la parte principal de este proyecto, es por ello por lo que en este punto lo vamos a dedicar a describir las partes del código y el uso gráfico a nuestra imagen al igual que describir la parte matemática.

Para empezar, recorrer la matriz de dos dimensiones que compone nuestra imagen se hace con dos bucles *for* que usan el alto total y el ancho total de la *imagen principal* como límites.

Para la realización del código no hemos utilizado la fórmula extendida, ya que generaba problemas, hemos usado una fórmula que se adapta a cada tramo de la circunferencia partiéndola en cuatro, afrontando cada tramo con exclusividad.

```
for (col = 1; col <= altoImg1; col++) {  
    for (row = 1; row <= anchoImg1; row++) {  
        imagen2(row, col) = imagen1(row,col);  
    }  
}
```

Código no funcional para mostrar la estructura de los dos bucles for.

7.1. Rotación de 0° a 90°

Una vez conocida la estructura describimos el primer tramo del funcionamiento.

El primer problema planteado es recorrer los grados desde 0 a 90°. Para obtener esta rotación hemos calculado mediante trigonometría el desplazamiento máximo que se tendría que usar en la imagen, en este tramo, vemos que el desplazamiento máximo es el ancho de la imagen1 que es el caso de la figura siguiente:

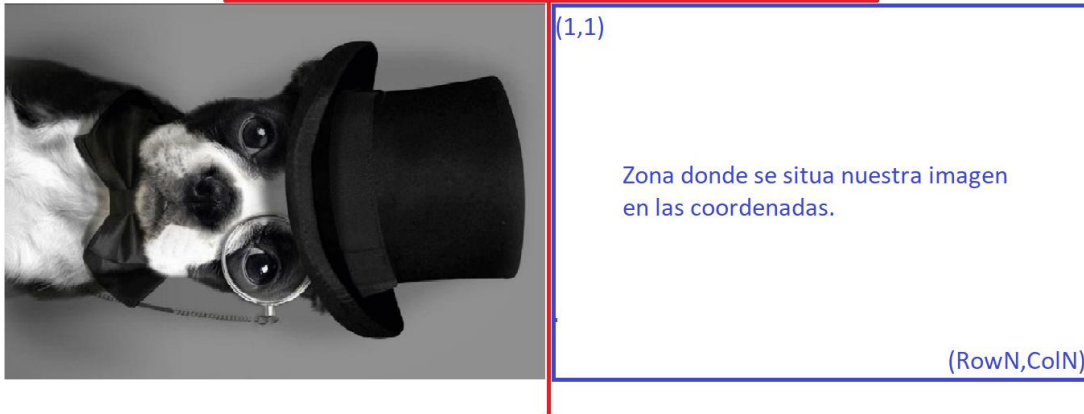


Imagen rotada 90°.

Como la imagen calculada por nuestra fórmula que si recordamos era:

$$X' = x \cdot \cos \Theta + y \cdot \sin \Theta;$$

$$Y' = -x \cdot \sin \Theta + y \cdot \cos \Theta;$$

Entonces para que nuestra imagen rotada esté en el sitio tenemos que desplazarla un total de el ancho de la imagen original, por tanto, nuestro código se queda como:

$$X' = x \cdot \cos \Theta + y \cdot \sin \Theta;$$

$$Y' = -x \cdot \sin \Theta + y \cdot \cos \Theta + \text{anchoImagen1};$$

Pero esta fórmula solo nos vale para calcular cuando nuestro ángulo $\Theta = 90^\circ$, para calcularlo con todo el rango de 0 a 90 grados necesitamos hacer lo siguiente:

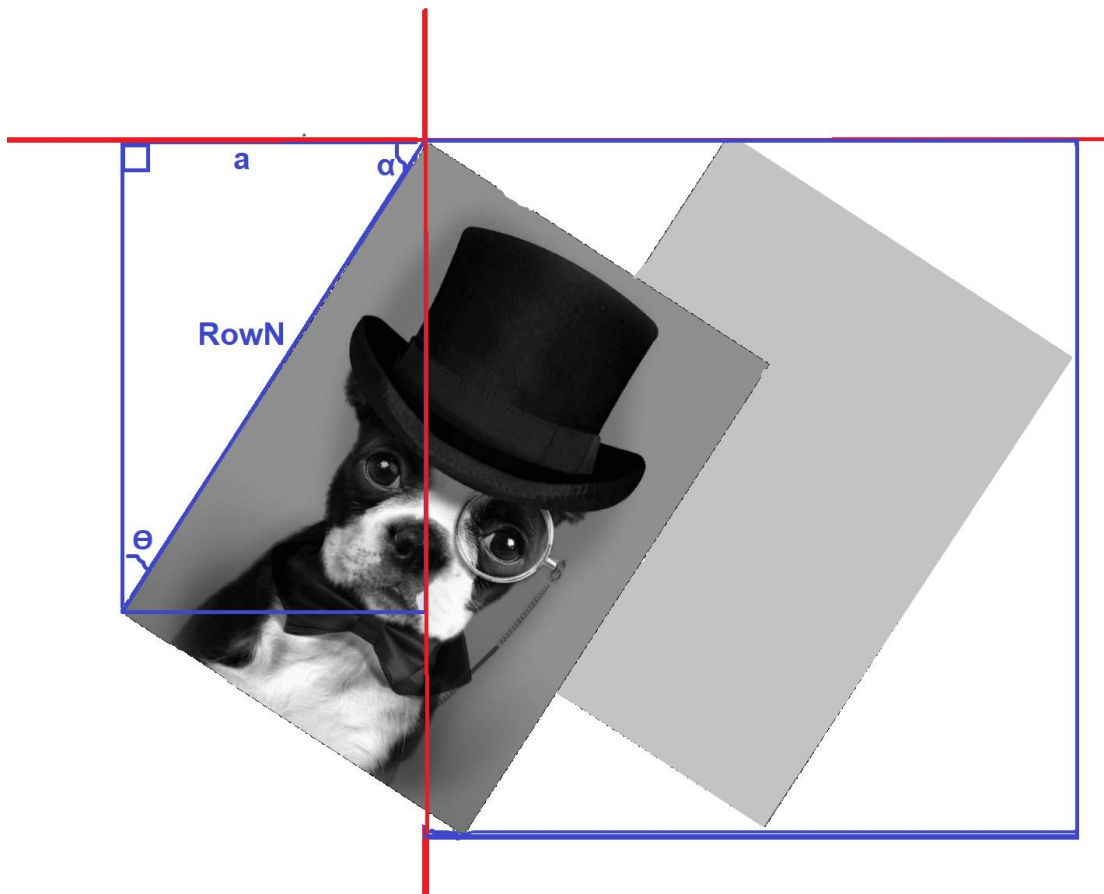


Imagen rotada 33°

$$a = \cos \Theta * \text{rowN}$$

Por tanto, la fórmula quedaría así:

$$X' = x * \cos \Theta + y * \sin \Theta;$$

$$Y' = -x * \sin \Theta + y * \cos \Theta + \cos \Theta * \text{anchoImagen1};$$

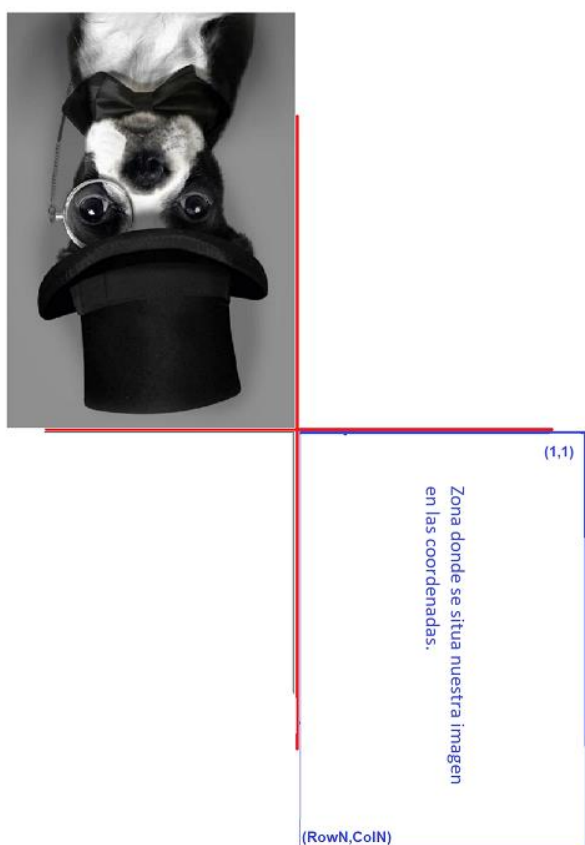
Y en el código:

```
for (col = 1; col <= altoImg1; col++) {
    for (row = 1; row <= anchoImg1; row++) {
        New_Row = (col * sin(angulo) + (row * cost));
        New_Col = col * cos(angulo) - row * sint + sin(angulo) * anchoImg1;
        int newRow = New_Row;
        int newCol = New_Col;
        imagen2(newRow, newCol) = imagen1(row, col);
    }
}
```


Un detalle que no hemos comentado es que el código contiene **int NewRow** e **int NewCol**, estas dos variables se utilizan para quitar la parte decimal de nuestra operación, ya que la matriz que es `imagen2` no acepta una coordenada con decimales y daría errores, pero esto hace que nuestra matriz tenga posiciones que no se llenen, por lo tanto, se quedan en negro, de esto hablaremos en el siguiente punto, pero el fenómeno se llama *aliasing*.

En el código de este informe aparecerá siempre el ángulo en **grados** para facilitar la visualización, pero en el código real tiene que ir en **radianes**

7.2. Rotación de 91° a 180°.



Al igual que en el tramo anterior, en este también tenemos que realizar un desplazamiento, pero en este tenemos que realizarlo a parte de realizarlo en el eje x , también tenemos que realizarlo en el eje y , la fórmula para 180° se nos queda así:

$$X' = x * \cos(\text{ángulo}) + y * \sin(\text{ángulo}) + \text{anchoImagen1};$$

$$Y' = -x * \sin(\text{ángulo}) + y * \cos(\text{ángulo}) + \text{altoImagen1};$$

Y para el cálculo de los demás ángulos del rango, la imagen se queda así:

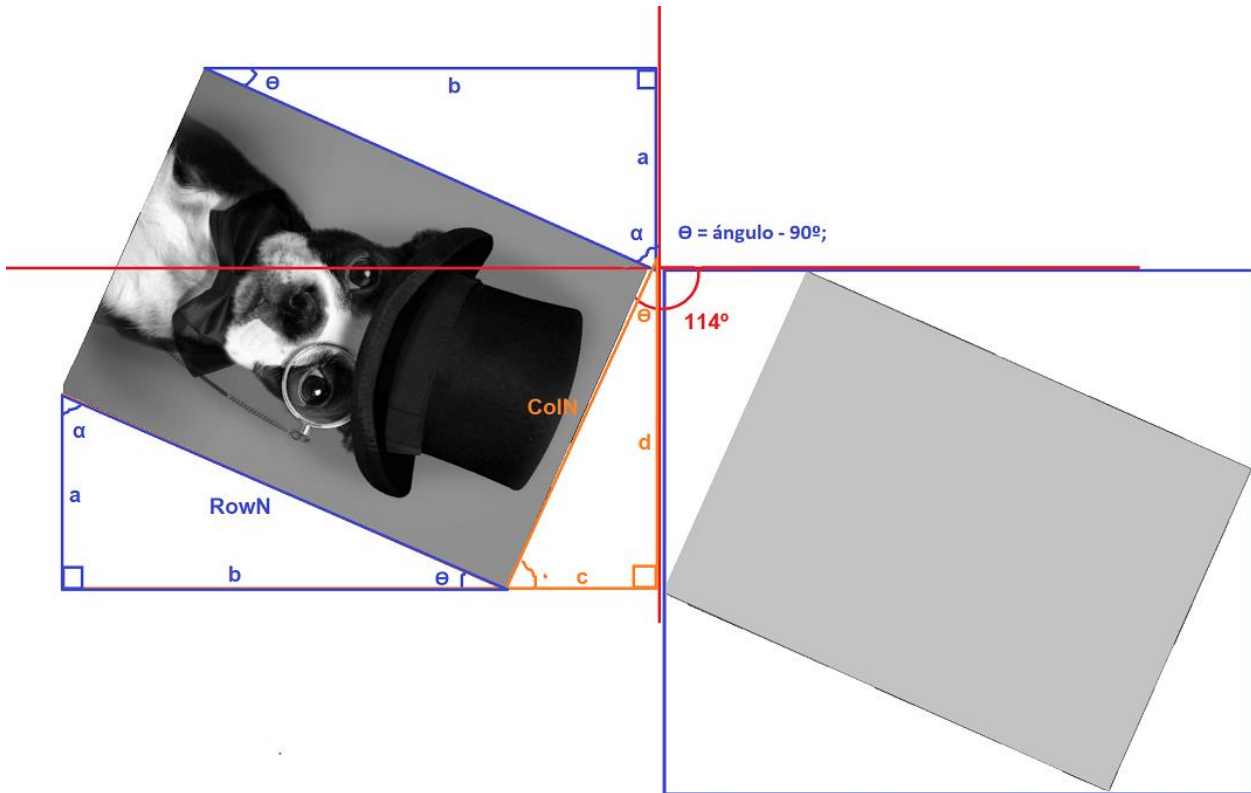


Imagen rotada 114°

$$b = \cos \Theta * \text{RowN};$$

$$c = \sin \Theta * \text{ColN};$$

$$a = \sin \Theta * \text{RowN};$$

Por tanto, la fórmula se queda así:

$$X' = x * \cos(\text{ángulo}) + y * \sin(\text{ángulo}) + \sin \Theta * \text{anchoImagen1};$$

$$Y' = -x * \sin(\text{ángulo}) + y * \cos(\text{ángulo}) + \sin \Theta * \text{altoImagen1} + \cos \Theta * \text{anchoImagen1};$$

El código en C++ se queda así:

```
for (col = 1; col <= altoImg1; col++) {
    for (row = 1; row <= anchoImg1; row++) {
        New_Row = (col * sin(angulo) + sin(angulo-90)*anchoImg1;
        New_Col = col * cos(angulo) - row * sint + sin(angulo-90) * altoImg1 + cos(angulo-90)*anchoImg1;
        int newRow = New_Row;
        int newCol = New_Col;
        imagen2(newRow, newCol) = imagen1(row, col);
    }
}
```

7.3. Rotación 181° a 270°

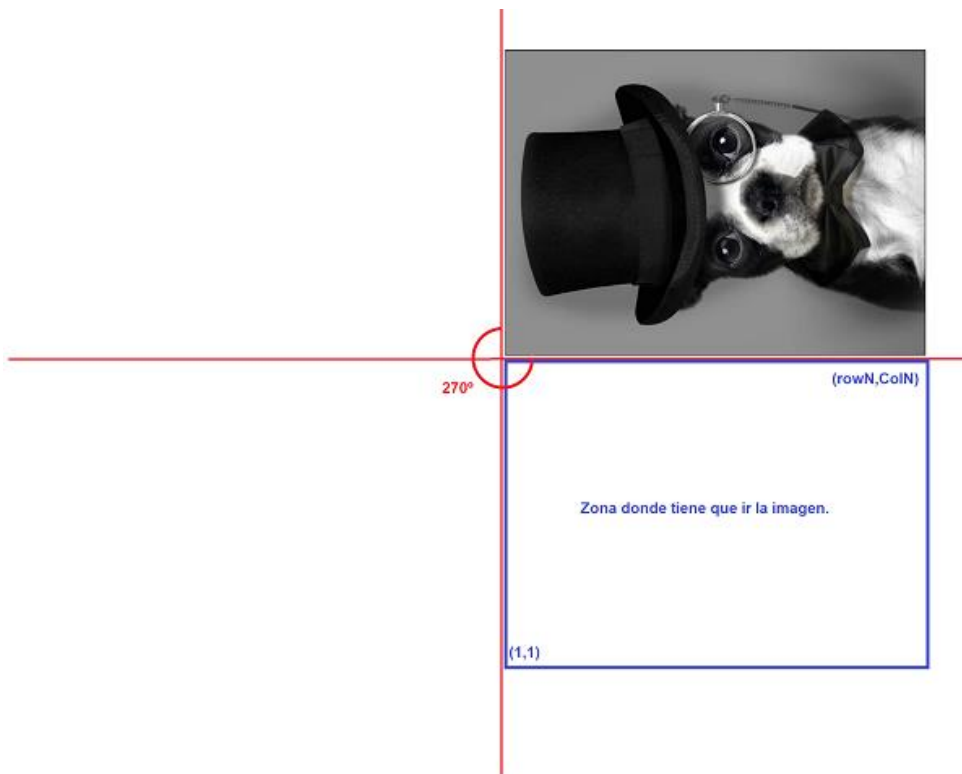


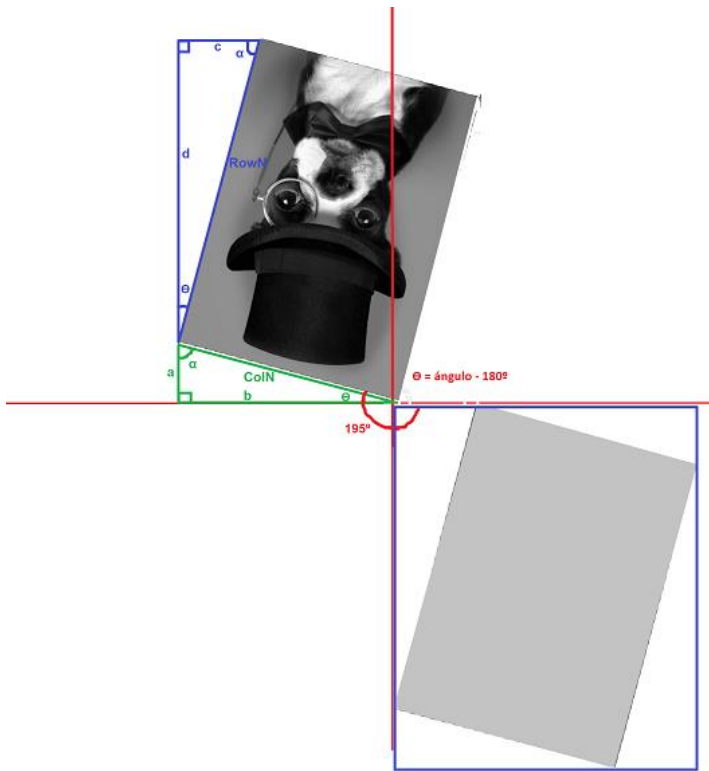
Imagen rotada 270°.

El tercer tramo es el de 181 a 270 grados, este tramo cuando está en 270 tiene que desplazarse el valor de ColN, por tanto, la fórmula queda así:

$$X' = x * \cos(\text{ángulo}) + y * \sin(\text{ángulo}) + \text{altoImagen1};$$

$$Y' = -x * \sin(\text{ángulo}) + y * \cos(\text{ángulo});$$

Pero como siempre, esta fórmula es solo válida para este caso de 270 grados, para el resto del rango de grados tenemos que crear una regla que va influida por la siguiente imagen:



$$a = \sin \Theta * \text{ColN};$$

$$b = \cos \Theta * \text{ColN};$$

$$d = \cos \Theta * \text{RowN};$$

$$\text{Ancho} = b = \cos \Theta * \text{ColN};$$

$$\text{Alto} = a+d = \sin \Theta * \text{ColN} + \cos \Theta * \text{RowN};$$

Imagen rotada 195°.

La fórmula queda de la siguiente forma:

$$X' = x * \cos(\text{ángulo}) + y * \sin(\text{ángulo}) + \sin \Theta * \text{altoImagen1} + \cos \Theta * \text{anchoImagen1};$$

$$Y' = -x * \sin(\text{ángulo}) + y * \cos(\text{ángulo}) + \cos \Theta * \text{altoImagen1};$$

El código en C++ queda de la siguiente forma:

```
for (col = 1; col <= altoImg1; col++) {
    for (row = 1; row <= anchoImg1; row++) {
        New_Row = (col * sin(angulo) + sin(angulo-180) * altoImg1 + cos(angulo-180)*anchoImg1;
        New_Col = col * cos(angulo) - row * sin(angulo) + cos(angulo-180)*altoImg1 ;
        int newRow = New_Row;
        int newCol = New_Col;
        imagen2(newRow, newCol) = imagen1(row, col);
    }
}
```

7.4. Rotación 271° a 360°

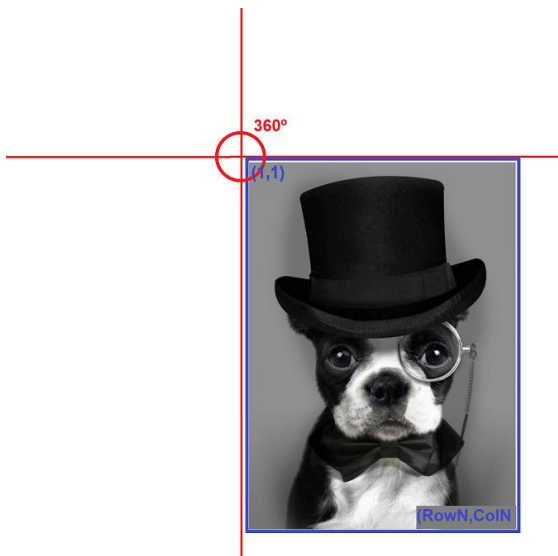
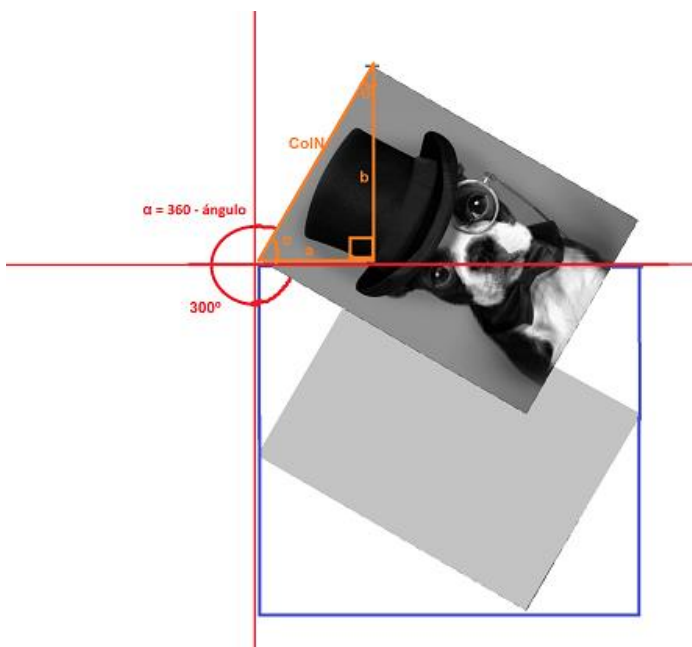


Imagen rotada 360°.

Para finalizar, el último tramo de rotaciones es de 271 a 360 grados, en el caso de 360° como es igual que 0° la fórmula quedaría de la siguiente forma ya que no necesita desplazamiento, aunque podamos no hacer nada ya que se queda en el mismo sitio que al empezar, tenemos que mostrar la fórmula que hace dar el giro completo a la imagen:

$$X' = x * \cos(360) + y * \sin(360);$$

$$Y' = -x * \sin(360) + y * \cos(360);$$



$$b = \sin \alpha * \text{ColN};$$

Imagen rotada 300°

Cuando la imagen es de 271 a 359 grados, el desplazamiento que se debe hacer viene definido por ColN, como muestra la figura de arriba, como en los casos anteriores, calculamos el desplazamiento mediante un triángulo rectángulo.

La fórmula queda de la siguiente forma:

$$X' = x * \cos(\text{ángulo}) + y * \sin(\text{ángulo}) + \sin(360 - \text{ángulo}) * \text{altoImg1};$$

$$Y' = -x * \sin(\text{ángulo}) + y * \cos(\text{ángulo});$$

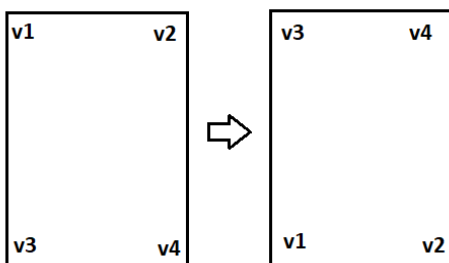
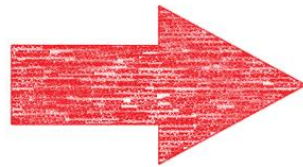
Y el código:

```
for (col = 1; col <= altoImg1; col++) {  
    for (row = 1; row <= anchoImg1; row++) {  
        New_Row = (col * sin(angulo) + sin(360 - angulo) * altoImg1;  
        New_Col = col * cos(angulo) - row * sin(angulo);  
        int newRow = New_Row;  
        int newCol = New_Col;  
        imagen2(newRow, newCol) = imagen1(row, col);  
    }  
}
```

8. Otras rotaciones.

A parte de las rotaciones sobre los ejes, tenemos un tipo de rotación más que también se llama Rotación espejo, o voltear la imagen. Existe la rotación espejo Vertical y la rotación espejo Horizontal.

8.1. Rotación espejo Vertical.

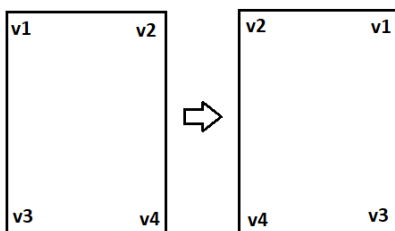
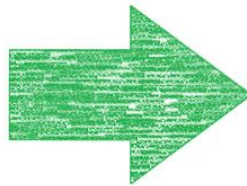


La rotación en espejo vertical lo que hace es invertir los píxeles de arriba por los píxeles de debajo de la imagen, haciendo que la imagen se quede como si se hubiese rotado 180° pero a la vez crea el efecto de estar mirándose en un espejo.

Código:

```
for (col = 1; col <= altoImg1; col++) {  
    for (row = 1; row <= anchoImg1; row++) {  
        imagen2(RowN - row + 1, newCol) = imagen1(row, col);  
    }  
}
```

8.2. Rotación espejo Horizontal.



El rotado espejo horizontal, el efecto que se consigue con él es el mismo que si te miraras en un espejo, lo conseguimos cambiando los píxeles de la izquierda por los de la derecha y viceversa.

Código:

```
for (col = 1; col <= altoImg1; col++) {  
    for (row = 1; row <= anchoImg1; row++) {  
        imagen2(row, ColN - col + 1) = imagen1(row, col);  
    }  
}
```

9. Anti – Aliasing.

El *Aliasing* es un efecto que se produce cuando se usa senos y cosenos para el procesamiento de imágenes porque cuando cogemos la imagen para encontrar su nueva posición hay veces que los resultados que nos da la operación tiene una serie de decimales que nos describen la posición exacta en la nueva imagen, pero las matrices no pueden procesar decimales, es por ello que estos decimales se desechan para dar números reales, y a causa de esto es posible que varias operaciones usen el mismo pixel y que haya otros píxeles que nunca se utilice, quedando píxeles negros. Los únicos grados que se libran de este fenómeno

son los ángulos de 0° , 90° , 180° , 270° , 360° ... porque estas operaciones van a tener siempre el seno y el coseno con valores -1,0 y 1 y recorrerán todas las posiciones.

Aquí tenemos ejemplos de *aliasing* en nuestra foto de ejemplo “SirPerro.bmp”.



Foto rotada 35°.

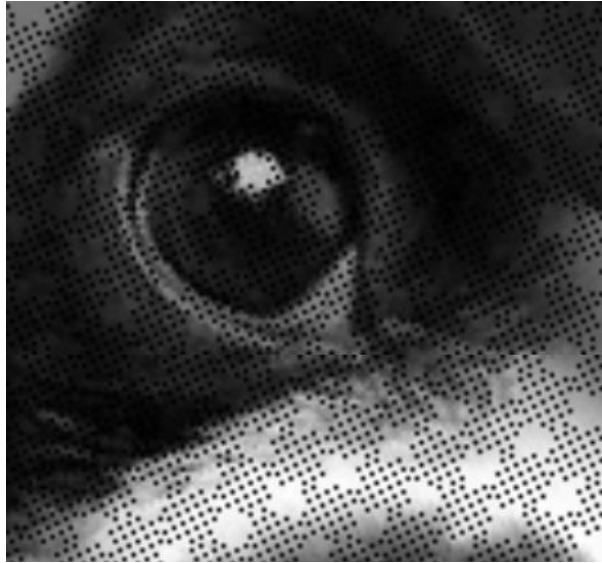


Foto rotada 45°

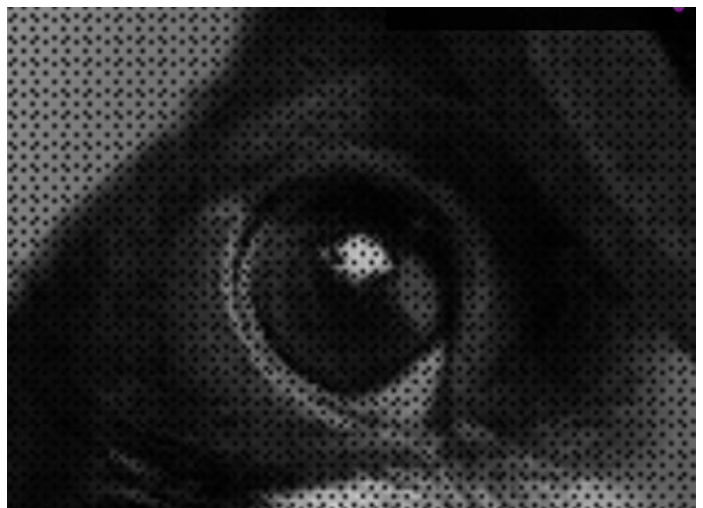
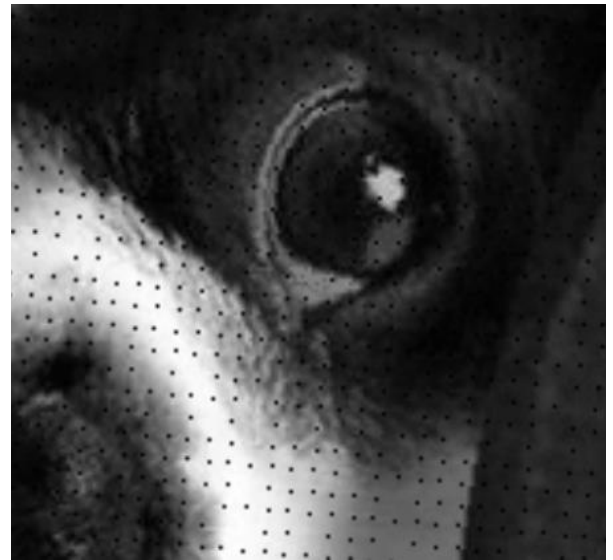




Foto rotada 100°



Esto supone un gran problema cuando procesamos las imágenes porque se produce una pérdida de información que a la larga es importante degradando nuestra imagen en cada acción. Es por ello por lo que existen formas de hacer que esta pérdida se note menos y tenga una solución.

Los métodos más utilizados son:

- Filtro de la media.
- Filtro de la mediana.
- Filtro de Gauss.
- Three Shears*

*Three Shears más que un filtro es una forma de mejorar el método de rotación rotando la imagen por secciones.

9.1. Filtro de la media.

El filtro de la media es un método de implementación para igualar los píxeles de una imagen, reduciendo la intensidad entre un píxel y sus vecinos.

La idea de utilizar este filtro es reemplazar los valores de cada píxel por la media aritmética de sus vecinos, incluyéndose.

Normalmente se utiliza la matriz de 3x3, pero se puede utilizar la de 5x5, aunque el resultado es más fuerte.

El resultado que da con nuestra imagen es la siguiente:

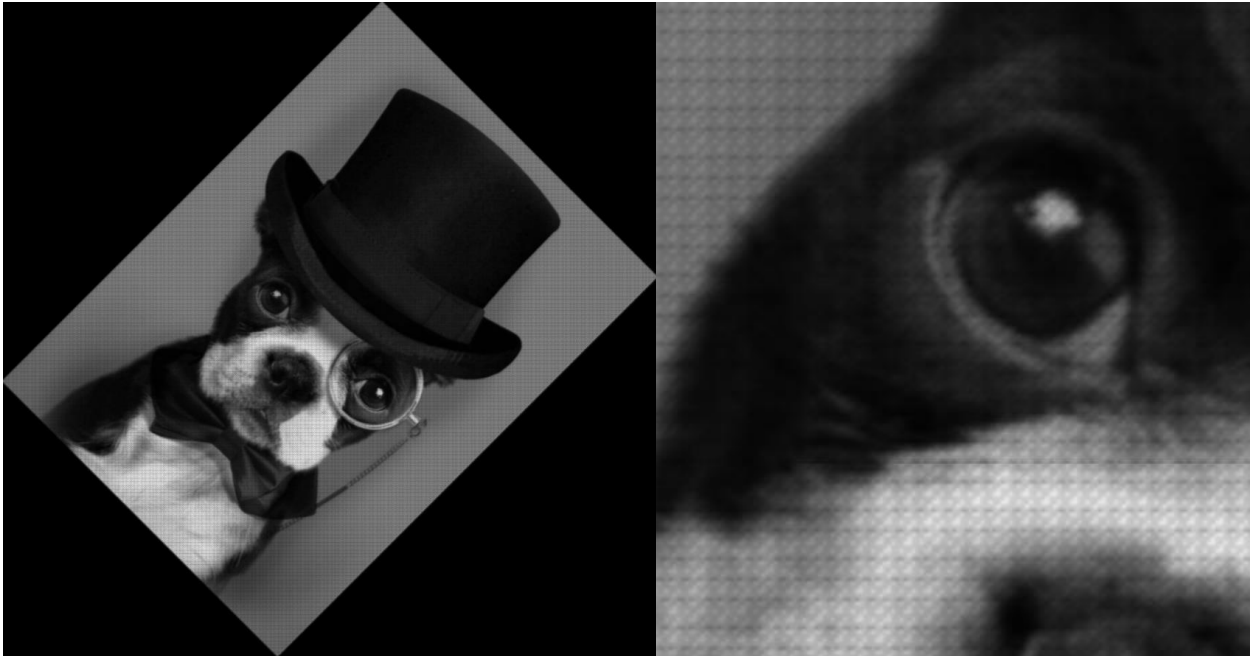


Foto rotada 45° Filtro media 3x3

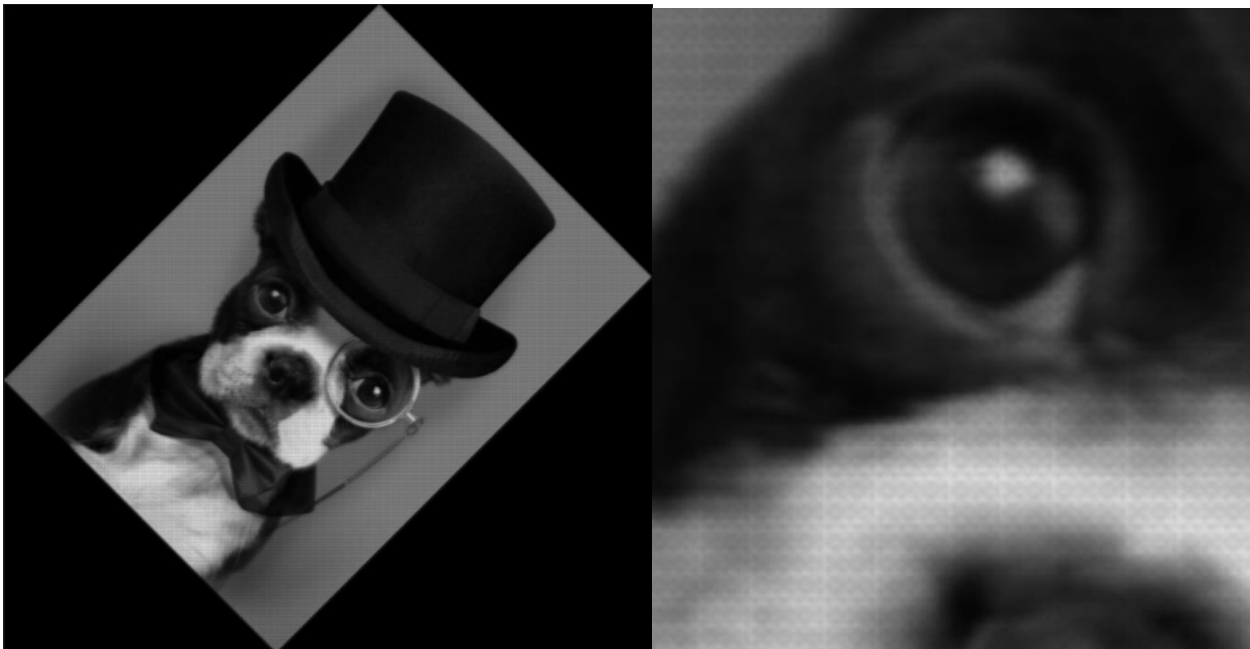


Foto rotada 45° Filtro media 5x5

Como vemos, este filtro arregla notablemente el problema que teníamos al rotar la imagen a 45°, pero a cambio de esto, la imagen se vuelve borrosa.

Código:

Matriz 3x3.

```
imagen3.MeanFilter(imagen2, 3);
```

Matriz 5x5.

```
imagen3.MeanFilter(imagen2, 5);
```

9.2. Filtro de la mediana.

El de la mediana es una mejora del filtro de la media, ya que el filtro de la media difumina bordes y otros detalles, cuando se quiere reducir más ruido que el que elimina el filtro de la media, utilizamos el de la mediana. Este filtro se centra en reemplazar el valor de gris de cada píxel por la mediana de los niveles de gris en un entorno de este píxel definido en función de su tamaño.



Foto rotada 45° Filtro de la mediana 3x3

Si nos fijamos, este método mejora muchísimo nuestra foto. Aunque le da un toque como si de una pintura se tratara. La versión de 7x7 no nos ayuda ya que al coger todos los vecinos próximos, nuestra imagen se oscurece.

Código:

```
imagen3.MedianFilter(imagen2, 3);
```

9.3. Filtro de Gauss.

El filtro de gauss es un operador de convolución de imágenes que se utiliza para emborronar imágenes y eliminar detalles y ruido.

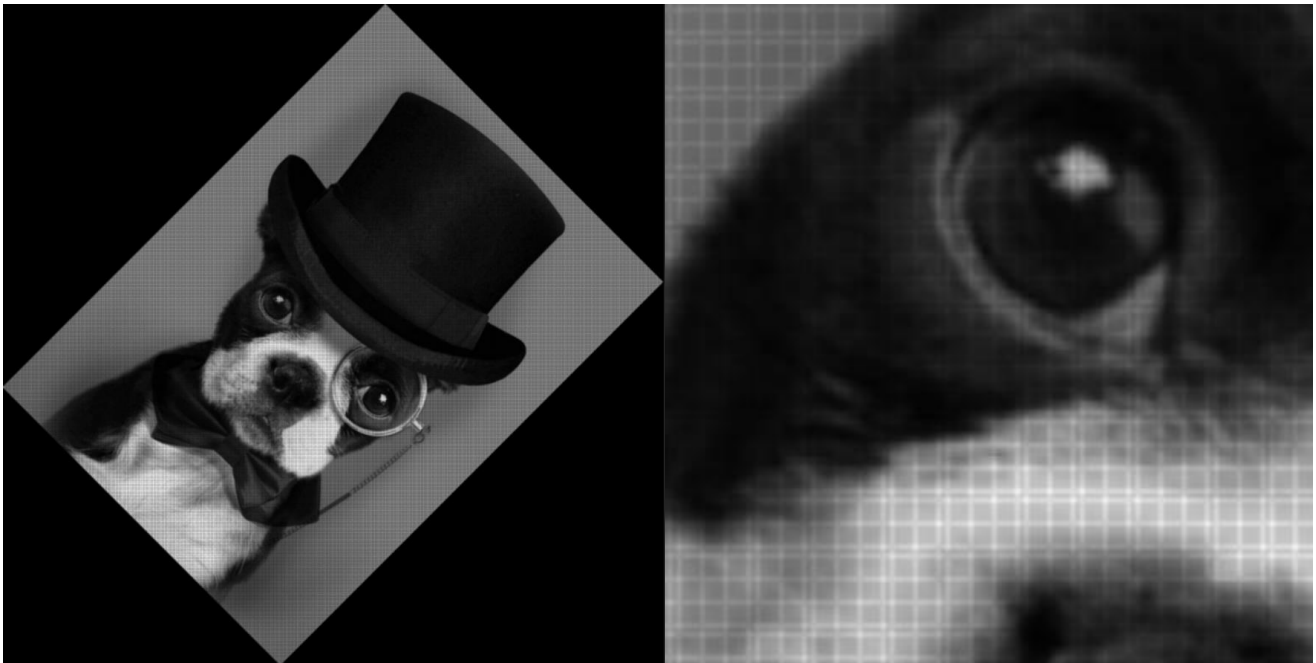


Foto rotada 45° Filtro de Gauss 3x3.



Foto rotada 45° Filtro Gauss 5x5

Si nos fijamos, el filtro de Gauss también soluciona parte de nuestro problema con el ruido de la imagen, pero nos deja unas rayas claras en toda la imagen.

Código:

Matriz 3x3.

```
C_Matrix gauss(1, 3, 1, 3);  
gauss.Gaussian(3);  
imagen3.CalculateConvolution(imagen3, gauss);
```

Matriz 5x5.

```
C_Matrix gauss(1, 5, 1, 5);  
gauss.Gaussian(5);  
imagen3.CalculateConvolution(imagen3, gauss);
```

9.4. Three Shears.

Este método se trata en realizar 3 pasos para lograr la rotación de una imagen en vez de rotarla de una vez como hemos hecho hasta ahora.

Este método trata en descomponer nuestra matriz de rotación en 3 matrices para realizar las operaciones por separado, estas matrices son las siguientes:

Por ejemplo, vamos a rotar la imagen SirPerro.bmp.

Escogemos el ángulo que queremos rotarlo, por ejemplo, 30°.

Ahora cogemos y hacemos un “Horizontal Shear” o lo que es lo mismo, escalamos la imagen un 30° y tratamos la imagen con la siguiente matriz:

$$\begin{vmatrix} 1 & -\tan(\Theta/2) \\ 0 & 1 \end{vmatrix}$$

$$\begin{vmatrix} 1 & 0 \\ \sin \Theta & 1 \end{vmatrix}$$



$$\begin{vmatrix} 1 & -\tan(\Theta/2) \\ 0 & 1 \end{vmatrix}$$



Resultado Three shears 30°



Resultado rotado C++ 30°

El resultado que nos dan estas imágenes sobre *Three Shears* no es real, porque no hemos implementado en nuestro código este método, por tanto, son imágenes simuladas, Pero la idea es hacer que nuestra imagen no tenga pérdidas porque utilizamos siempre matrices que nos van a dar resultados que vamos a poder controlar más la pérdida.

9.5 Solución Aliasing.

En los puntos anteriores hemos expuesto que la solución para evitar la pérdida es aplicar filtros, pero no hemos expuesto cuál ha sido nuestra solución ante este problema.

Nuestra solución ha sido aplicar el filtro de la mediana 3x3 que nos dejaba la imagen como si estuviese pintada, pero entonces, volvíamos a sobre escribir la imagen de nuevo, por tanto, los puntos que no se

utilizan en la operación se quedan del color que ha hecho la mediana, y los puntos que se usan, han sido escritos de nuevo como antes de aplicarse el filtro.

10. Ejemplo de ejecución en C++

Tras todos los demás puntos explicando como funciona el algoritmo, los problemas planteados hay que ver como se ha implementado el código en C++.

Todo el código para la rotación se alberga en TDI.cpp, dentro de este código tenemos 4 métodos. **rotacion**, **espejoVertical**, **espejoHorizontal** y **main**.

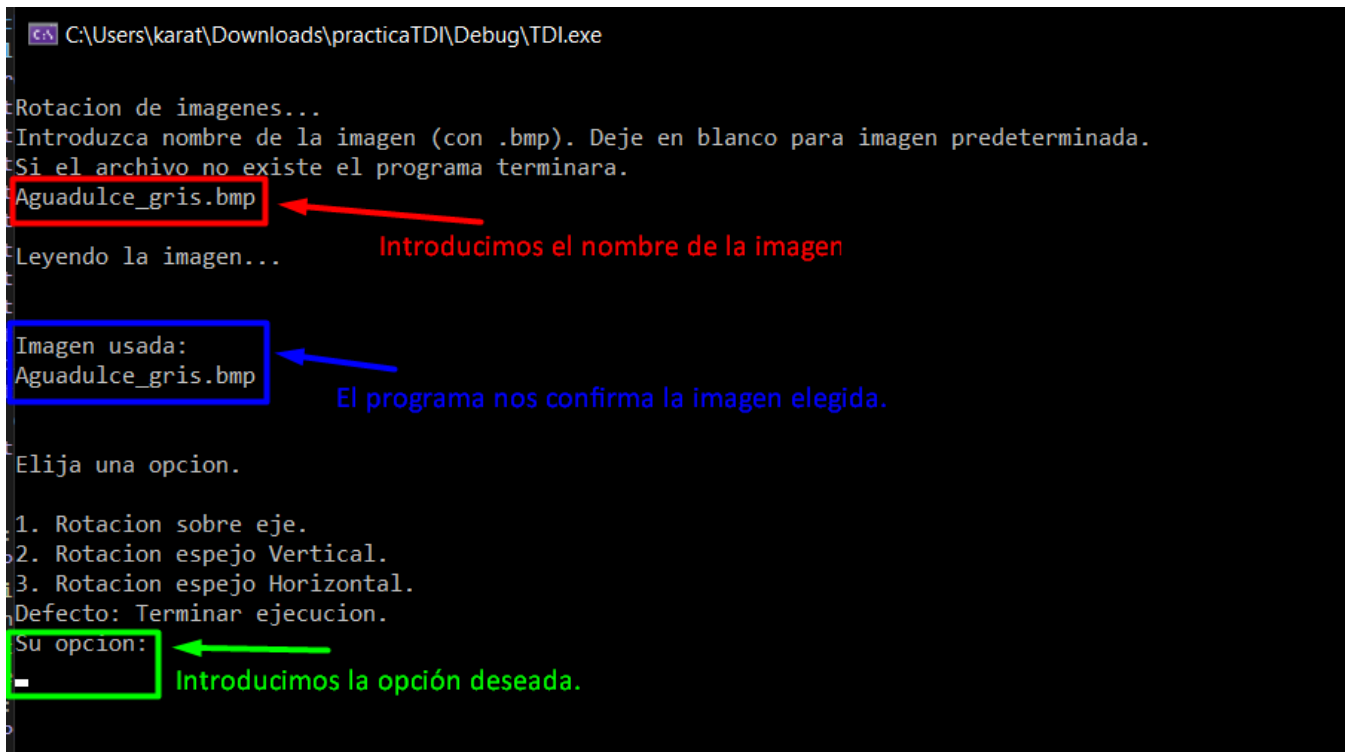
```
4  #include <C_Image.hpp>
5  #include <iostream>;
6  #include <cmath>;
7  #include <string>
8  #define _USE_MATH_DEFINES
9  using namespace std;
10 int Test(int argc, char **argv);
11
12 void rotacion(C_Image imagen1, double angulo, string str){ ... }
149 void espejoVertical(C_Image imagen1, string str){ ... }
164 void espejoHorizontal(C_Image imagen1, string str){ ... }
178
179
180
181
182 int main(int argc, char **argv){ ... }
```

Como su nombre indica, **rotacion** contiene todo el código que realiza las operaciones para rotar una imagen, para hacerlo funcionar necesitamos pasarle por parámetro una imagen que quiera ser rotada, un ángulo de rotación (funciona con decimales) y el nombre del archivo.

espejoVertical y **espejoHorizontal** son los métodos que nos harán la transformación de nuestra imagen a modo espejo en vertical y horizontal respectivamente, tan solo necesitan la **imagen** y el **String de nombre**.

10.1. Ejecución común.

Cuando ejecutamos este programa lo primero que nos pedirá es un nombre de un archivo con .bmp incluido. Si no introducimos ningún nombre, el programa se cerrará. Si hemos elegido un nombre que se encuentre en la carpeta **RUN** de nuestro proyecto entonces pondrá lo siguiente:



```
C:\Users\karat\Downloads\practicaTDI\Debug\TDI.exe

Rotacion de imagenes...
Introduzca nombre de la imagen (con .bmp). Deje en blanco para imagen predeterminada.
Si el archivo no existe el programa terminara.
Aguadulce_gris.bmp
Leyendo la imagen...
Imagen usada:
Aguadulce_gris.bmp
Elija una opcion.

1. Rotacion sobre eje.
2. Rotacion espejo Vertical.
3. Rotacion espejo Horizontal.
Defecto: Terminar ejecucion.
Su opcion:
1
```

Introducimos el nombre de la imagen

El programa nos confirma la imagen elegida.

Introducimos la opción deseada.

Dependiendo de la opción que elijamos nos iremos a rotación si elegimos el 1 o a espejo si elegimos el 2 y el 3.


10.2. Opción Rotación.

Cuando elijamos rotación lo primero que se nos va a pedir es el ángulo de rotación que queremos introducirle a nuestra imagen, permite introducir decimales.

Tras introducir el ángulo, ya no hará falta introducir nada más y solo hay que esperar a que el programa termine y se creará nuestra imagen en la carpeta **Run** con el nombre que nos dirá el programa (nombre de la imagen con el ángulo de rotación).

Mientras se ejecuta el código nos dirá el **tamaño de la imagen introducida**, el **tamaño de la nueva imagen** y si se está aplicando o no el **anti-aliasing** (filtro de la mediana)

```
Consola de depuración de Microsoft Visual Studio
2. Rotacion espejo Vertical.
3. Rotacion espejo Horizontal.
Defecto: Terminar ejecucion.
Su opcion:
1
Opcion elegida:
Rotacion de imagenes sobre su eje.
Seleccione el angulo de rotacion (en grados): 25 ← introducimos el ángulo
La imagen introducida tenia las siguientes características:
alto: : 629
ancho: : 381
La nueva imagen tendra las siguientes características:
Nuevo alto: : 731
Nuevo ancho: : 611
Aplicando filtro de la Mediana para eliminar el ruido... (Espere unos segundos)
La imagen ha sido rotada con exito...
Guardando...
Finalizado.
El nombre de la imagen es:
Aguadulce_gris_Rotada_25.000000.bmp ← Nombre de la nueva imagen
```




Aguadulce_gris_Rotada_25.000000.bmp

10.3. Opción espejo.

La opción espejo al elegirla no hará falta hacer nada más, se creará una imagen con el nombre de nuestra imagen y el tipo de rotación espejo que se ha elegido en la carpeta **Run**, y mientras se ejecuta el programa se nos mostrará el tamaño de la imagen.

```
1 Elija una opcion.
2
3 1. Rotacion sobre eje.
4 2. Rotacion espejo Vertical.
5 3. Rotacion espejo Horizontal.
6 Defecto: Terminar ejecucion.
7 Su opcion:
8 2
9 Opcion elegida:
10 Modo espejo Vertical.
11 La imagen introducida tiene las siguientes características:
12
13 alto: : 629
14 ancho: : 381
15 Guardando.
16
17 Finalizado.
18 El nombre de la imagen es:
19 Aguadulce_gris_EspejoVertical.bmp ← Nombre de la nueva imagen
```



Aguadulce_Gris_EspejoVertical.bmp

11. Conclusión.

Para finalizar, en este proyecto se ha llevado una imagen desde una posición a otra, solucionando todos los problemas que han ido surgiendo, lo más difícil ha sido calcular todos los aspectos de la imagen y encontrar el método óptimo para solucionar el aliasing, como resultado de todo, tenemos imágenes finales nítidas que tienen su tamaño real y un añadido que es lo que hace que estas imágenes .bmp sigan siendo rectangulares pese a estar girada en cualquier ángulo posible. Como futuras implementaciones queda usar *three shears* como método para evitar el uso del filtro de la mediana para arreglar el aliasing de la imagen y que la ejecución fuera más rápida. Otro de los aspectos a considerar sería el uso de un algoritmo que nos detectara donde empieza la foto y donde terminan los bordes añadidos artificialmente para que si queremos rotar la foto de nuevo esta no se haga más gruesa con estos bordes.

12. Bibliografía.

Guindos, F., & Piedra, J. A. (2001). *TRATAMIENTO DIGITAL DE IMAGENES CON IMtdi*. Almería, España: pdf.

BMP converter. (s. f.). Recuperado 24 de mayo de 2020, de <https://online-converting.com/image/convert2bmp/#>

Formula for rotating a vector in 2D. (s. f.). Recuperado 31 de marzo de 2020, de https://matthew-brett.github.io/teaching/rotation_2d.html

Matriz de rotación. (2020, abril 8). Recuperado 10 de abril de 2020, de https://es.wikipedia.org/wiki/Matriz_de_rotaci%C3%B3n

Rotating Images. (s. f.). Recuperado 9 de marzo de 2020, de <http://datagenetics.com/blog/august32013/index.html>

Turnero, P. (s. f.). Procesamiento de imágenes (Traslación, Escala, Rotación, Inclinación). Recuperado 10 de marzo de 2020, de <https://www.monografias.com/trabajos108/procesamiento-imagenes-traslacion-escala-rotacion-inclinacion/procesamiento-imagenes-traslacion-escala-rotacion-inclinacion.shtml>