Informática Gráfica

Silvia Perez Ruiz
E-mail: silvia.perezr@um.es, DNI: 49215974E
Raúl Hernández Martínez
E-mail: raul.hernandezm@um.es, DNI: 24423648V

Práctica Informática Gráfica.

Grado en Ingeniería Informática
Informática Gráfica

UNIVERSIDAD DE
MURCIA

Índice

1 Rotar imagen y reescalar. 2

2 Copiar al portapapeles 6

3 Ver información 8

4 Bloquear pincel 11

5 Nueva desde el portapapeles 12

6 Convertir a color falso 14

7 Modelos de color 18

8 Morfología matemática 21

1 Rotar imagen y reescalar.

Hemos llevado a cabo la creación de un fichero nuevo llamado **rotar_imagen.cpp** con su correspondiente **rotar_imagen.h** y **rotar_imagen.ui**. Además, hemos añadido al fichero **imagenes.cpp** e **imagenes.h** la función **ver_rotar_escalar_imagen** para poder ver como se realiza la rotación de la imagen con la escala seleccionada.

En dicha función vemos que tenemos como argumentos de entrada aquellos parámetros para tratar la foto actual y donde mostrar la foto resultante, junto a los parámetros del ángulo de la rotación de la imagen, la escala requerida y un booleano para saber si queremos guardar la imagen.

Hacemos uso de la función **rotar_angulo** ya creada en el fichero **imagenes.cpp** para poder realizar la accion. A dicho metodo le incorporamos los parámetros nombrados anteriormente.

```
1
2  void ver_rotar_escalar_imagen(int nfoto, int nres, double angulo, double escala, bool
       guardar)
3  {
4      assert(nfoto >= 0 && nfoto < MAX_VENTANAS && foto[nfoto].usada);
5      Mat entrada = foto[nfoto].img;
6      Mat imgRotadaEscalar;
7      rotar_angulo(entrada, imgRotadaEscalar, angulo, escala, 1);
8      if (guardar) {
9          crear_nueva(nres, imgRotadaEscalar);
10     }
11     else {
12         imshow("RotarEscalar", imgRotadaEscalar);
13     }
14 }
```

Listing 1: ver_rotar_escalar_imagen.

```
1
2  void rotar_angulo (Mat imagen, Mat &imgRes, double angulo, double escala, int modo)
3  {
4      double w= imagen.size().width, h= imagen.size().height;
5      Size sres;
6      if (modo==0) {     // Reescalar con proporci n al original
7          sres.width= int(w*escala);
8          sres.height= int(h*escala);
9      }
10     else if (modo==1)  // Reescalar y ampliar para caber entera
11         sres.width= sres.height= int(sqrt(w*w + h*h)*escala);
12     else               // Reescalar y recortar para no mostrar borde
13         sres.width= sres.height= int(min(w, h)*escala/sqrt(2.0));
14     imgRes.create(sres, imagen.type());
15     double sa= sin(angulo*M_PI/180), ca= cos(angulo*M_PI/180);
16     double cx= -w/2*ca-h/2*sa, cy= w/2*sa-h/2*ca;
17     Mat M= getRotationMatrix2D(Point2f(0,0), angulo, escala);
18     M.at<double>(0,2)= sres.width/2+cx*escala;
19     M.at<double>(1,2)= sres.height/2+cy*escala;
20     imgRes= color_pincel;
21     warpAffine(imagen, imgRes, M, sres, INTER_CUBIC);
22 }
```

Listing 2: ver_rotar_angulo.

A continuación mostramos los ficheros **rotar_imagen.cpp** y **rotar_imagen.h**. Mostramos que en el constructor de rotar_imagen le añadimos los parámetros para obtener la foto actual, y como argumentos a la clase, uno para los grados, inicializado a 0 y otro para la escala inicializado a 1.

Tenemos que destacar, que hemos acotado una escala máxima de 6.0 debido a que, después de realizar varias pruebas, nos hemos dado cuenta que a partir de 7.0 llega a consumir mucha cantidad significativa de memoria, ya que estaríamos escalando imágenes significativamente grandes y el programa no lo soportaría.

Disponemos de dos spinBox y uno de ellos sincronizado con un horizontalSlider para que el usuario pueda seleccionar a su gusto el grado de rotación mientras que ve como rota la imagen y la escala.

```
1
2  #ifndef ROTAR_IMAGEN_H
3  #define ROTAR_IMAGEN_H
4
5  #include <QDialog>
6
7  namespace Ui {
8  class Rotar_imagen;
9  }
10
11 class Rotar_imagen : public QDialog
12 {
13     Q_OBJECT
14
15 public:
16     explicit Rotar_imagen(int numfoto, int numres, QWidget *parent = nullptr);
17     ~Rotar_imagen();
18
19 private slots:
20     void on_horizontalSlider_valueChanged(int value);
21
22     void on_spinBox_valueChanged(int arg1);
23
24     void on_spinBox_2_valueChanged(int arg1);
25
26     void on_Rotar_imagen_accepted();
27
28     void on_Rotar_imagen_rejected();
29
30 private:
31     Ui::Rotar_imagen *ui;
32     int grado;
33     int escala;
34     int nfoto, nres;
35 };
36
37 #endif // ROTAR_IMAGEN_H
```

Listing 3: rotar_imagen.h.

```
1
2  #include "rotar_imagen.h"
3  #include "ui_rotar_imagen.h"
4
5  #include "imagenes.h"
6
7  Rotar_imagen::Rotar_imagen(int numfoto, int numres, QWidget *parent) :
8      QDialog(parent),
9      ui(new Ui::Rotar_imagen)
10 {
11     ui->setupUi(this);
12     nfoto = numfoto;
13     nres = numres;
14     escala = 1;
15     grado = 0;
16     ver_rotar_escalar_imagen(nfoto, nres, grado, escala);
17 }
18
19 Rotar_imagen::~Rotar_imagen()
20 {
```

```
21      delete ui;
22  }
23
24  void Rotar_imagen::on_horizontalSlider_valueChanged(int value)
25  {
26      ui->spinBox->setValue(value);
27      grado = value;
28      ver_rotar_escalar_imagen(nfoto, nres, grado, escala);
29  }
30
31  void Rotar_imagen::on_spinBox_valueChanged(int arg1)
32  {
33      ui->horizontalSlider->setValue(arg1);
34      grado = arg1;
35      ver_rotar_escalar_imagen(nfoto, nres, grado, escala);
36  }
37
38  void Rotar_imagen::on_spinBox_2_valueChanged(int arg1)
39  {
40      escala = arg1;
41      ver_rotar_escalar_imagen(nfoto, nres, grado, escala);
42  }
43
44  void Rotar_imagen::on_Rotar_imagen_accepted()
45  {
46      ver_rotar_escalar_imagen(nfoto, nres, grado, escala, true);
47      destroyWindow("RotarEscalar");
48  }
49
50  void Rotar_imagen::on_Rotar_imagen_rejected()
51  {
52      mostrar(nfoto);
53      destroyWindow("RotarEscalar");
54  }
```

Listing 4: rotar_imagen.cpp.

Ejemplos del opcional de Rotar imagen y reescalar.

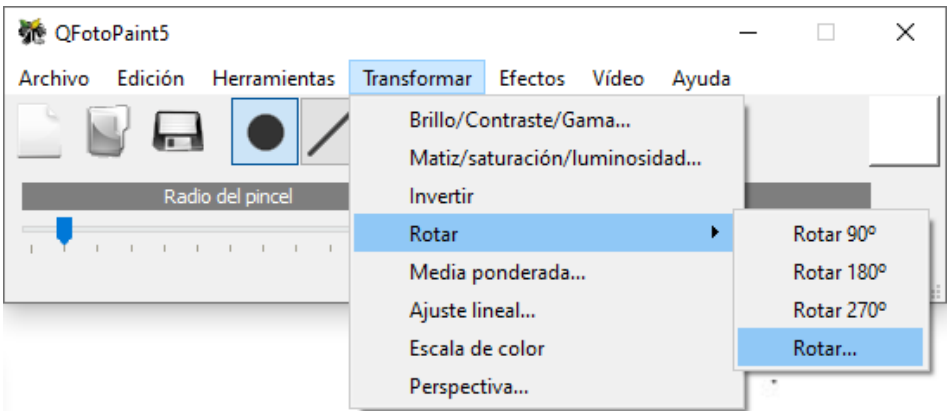

Figura 1: Menú desplegable para rotar imagen y reescalar.

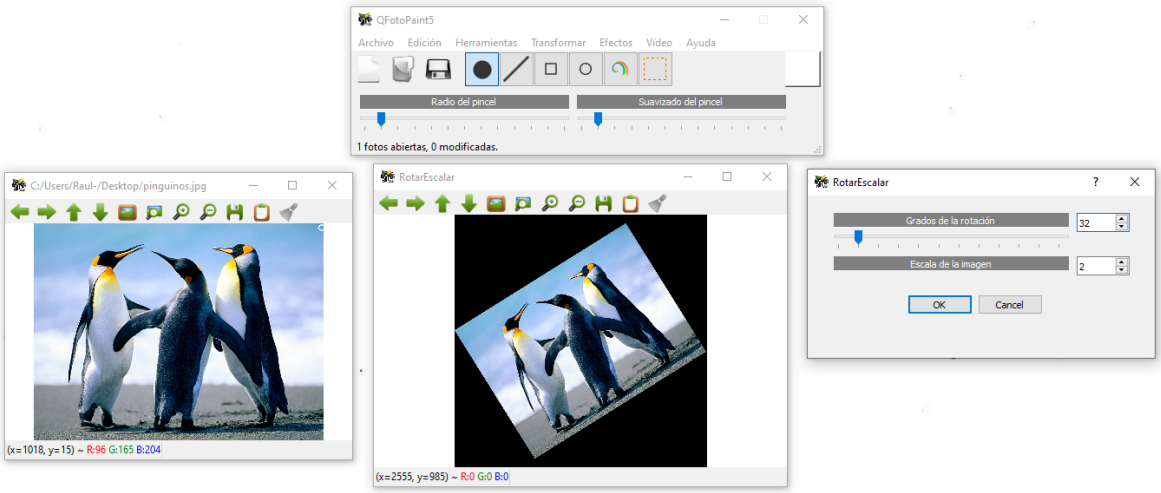

Figura 2: Ejemplo 1 de rotación de imagen y escala.

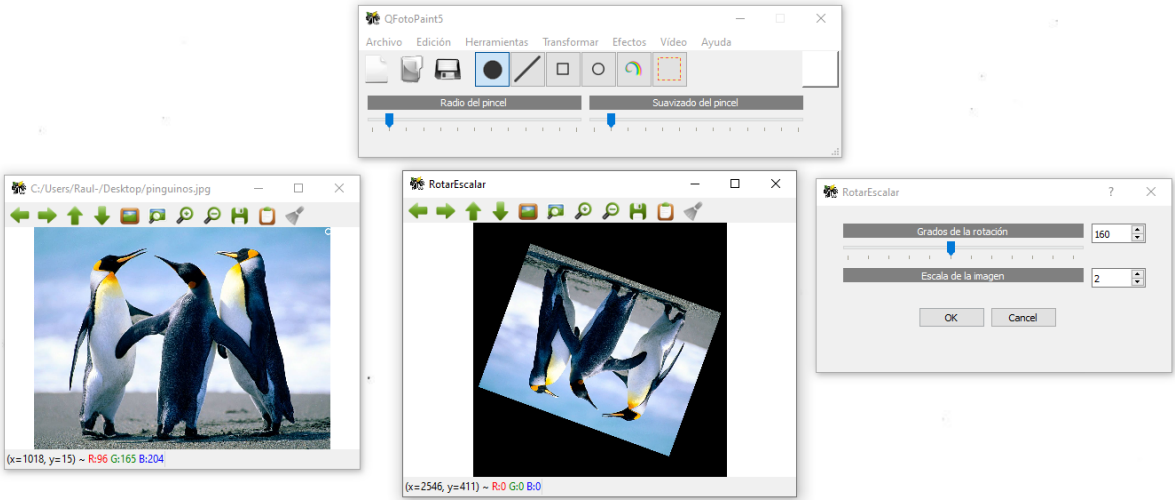

Figura 3: Ejemplo 2 de rotación de imagen y escala.

2 Copiar al portapapeles

Para este opcional no hacia falta crear una nueva clase, para poder añadirlo lo único que hemos tenido que hacer es irnos al fichero **mainwindow.ui**, y en la pestaña de `Edicion` añadimos un nuevo campo que se llama `Copiar a portapapeles`.

Una vez hecho eso, tenemos que irnos al **mainwindow.cpp** e implementar la función triggered para poder llamar la implementación del opcional que se encuentra en **imagenes.cpp**. La función triggered se queda asi:

```
1
2  void MainWindow::on_actionCopiar_a_portapapeles_triggered()
3  {
4      if (foto_activa() != -1)
5          copiar_a_portapapeles(foto_activa());
6  }
```

Listing 5: Función triggered().

La siguiente función que nos queda es la que consigue copiar la imagen al portapapeles. Primero nos guardamos una copia de la foto que esta activa en este momento. Cambiamos el color para poder adaptarlo al formato de la QImage. Creamos el QClipboard y a continuación creamos la QImage con los datos que tenemos de la imagen original. Cambiamos el formato otra vez y asignamos la imagen al portapapeles general.

```
1
2  void copiar_a_portapapeles(int nfoto)
3  {
4      Mat copia = foto[nfoto].img(foto[nfoto].roi).clone();
5
6      cvtColor(copia, copia, COLOR_BGR2RGB);
7      QClipboard *clipboard = QGuiApplication::clipboard();
8      QImage qImage(copia.data, copia.cols, copia.rows, copia.step, QImage::Format_RGB888);
9      qImage = qImage.convertToFormat(QImage::Format_ARGB32);
10     clipboard->setImage(qImage, QClipboard::Clipboard);
11
12 }
```

Listing 6: Función copiar_a_portapapeles.

Con esto ya tendríamos hecho la implementación de copiar al portapapeles.

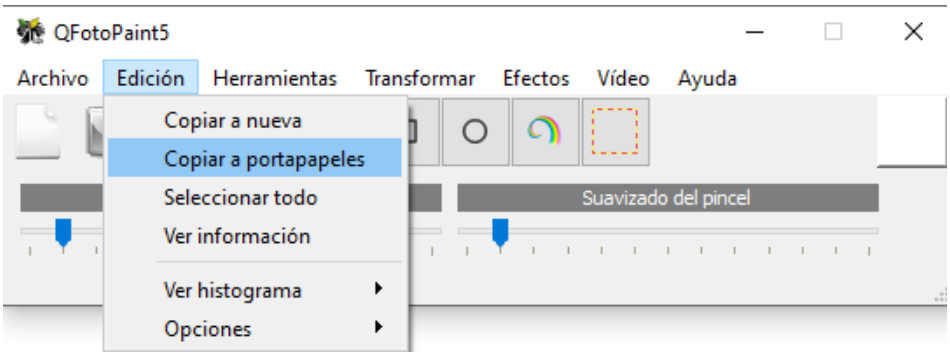

Figura 4: Menú desplegable para copiar al portapapeles.

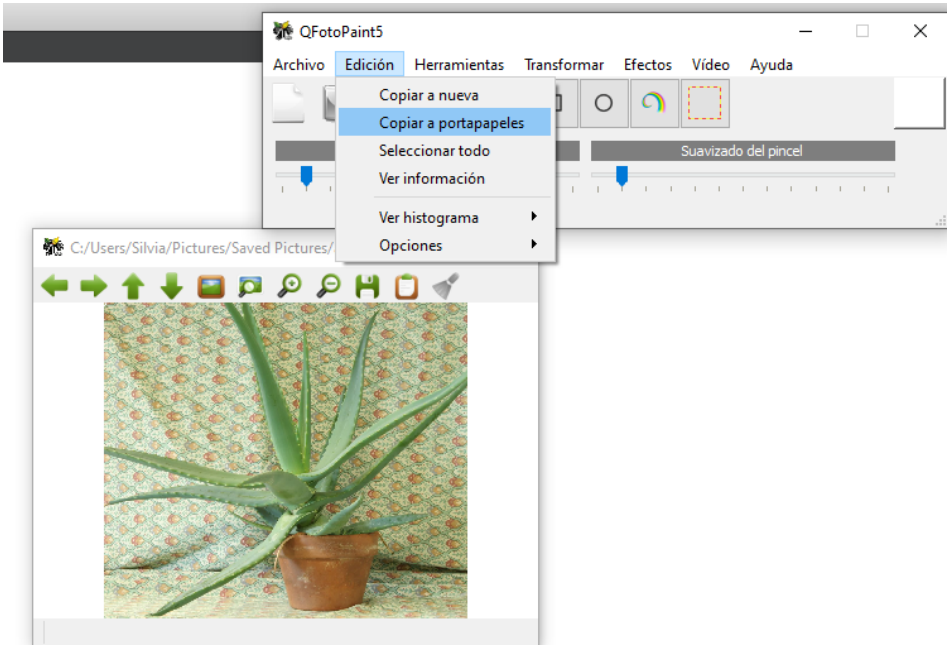

Figura 5: Copiar la imagen activa.

Figura 6: Pegar la imagen copiada.

3 Ver información

Para poder realizar este opcional, si que hemos tenido que crear una nueva clase llamada **ver_info**. Lo primero que hemos hecho ha sido crear la ventana de **ver_info.ui** donde esta llena de labels de los cuales, algunos se tendrán que rellenar con la información de la imagen activa.

Después hemos ido al fichero **mainwindow.cpp** y hemos puesto en el menú edición una nueva opción que se llama `Ver infomacin`. A continuación hemos implementado la función triggered para esa opción para que nos cree un constructor de la nueva clase.

```
1
2  void MainWindow::on_actionVer_informaci_n_triggered()
3  {
4      if (foto_activa() != -1) {
5          ver_info vi(foto_activa());
6          vi.exec();
7      }
8  }
```

Listing 7: Función triggered().

Esta función nos lleva hasta el constructor de **ver_info.cpp**. Donde guardará en la la imagen que devuelva la función `ver\_informacin(int nfoto)` implementada en **imagenes.cpp**. Una vez recuperada la imagen, llamamos a una función de la clase **ver_info.cpp** que se llama `setInfo()` donde establecemos en cada label la información de la imagen recuperada.

Código clase **ver_info.h**:

```
1
2  #ifndef VER_INFO_H
3  #define VER_INFO_H
4
5  #include <QDialog>
6  #include <opencv2/opencv.hpp>
7  using namespace cv;
8  #include "imagenes.h"
9
10 namespace Ui {
11 class ver_info;
12 }
13
14 class ver_info : public QDialog
15 {
16     Q_OBJECT
17
18 public:
19     explicit ver_info(int num_foto, QWidget *parent = nullptr);
20     ~ver_info();
21     void setInfo();
22
23 private:
24     Ui::ver_info *ui;
25     int nfoto;
26     Mat img;
27 };
28
29 #endif // VER_INFO_H
```

Listing 8: ver_info.h.

Código clase **ver_info.cpp**:

```
1
2  #include "ver_info.h"
3  #include "ui_ver_info.h"
4
```

```
ver_info::ver_info(int num_foto, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::ver_info)
{
    ui->setupUi(this);
    nfoto = num_foto;
    img = ver_informacion(nfoto);
    setInfo();
}

ver_info::~ver_info()
{
    delete ui;
}

void ver_info::setInfo()
{

    int ancho = img.cols;
    int alto = img.rows;
    int profundidad = img.elemSize() * 8;
    int canales = img.channels();
    int memoria = img.total() * img.elemSize();
    Scalar meanColor = mean(img);
    int meanRed = cvRound(meanColor[2]);
    int meanGreen = cvRound(meanColor[1]);
    int meanBlue = cvRound(meanColor[0]);

    QColor meanColorQt(meanRed, meanGreen, meanBlue);

    ui->label_3->setNum(ancho);
    ui->label_18->setNum(alto);
    ui->label_5->setNum(profundidad);
    ui->label_7->setNum(canales);
    ui->label_9->setNum(memoria);
    ui->label_12->setNum(meanRed);
    ui->label_14->setNum(meanGreen);
    ui->label_16->setNum(meanBlue);
    ui->label_19->setAutoFillBackground(true);
    ui->label_19->setPalette(QPalette(meanColorQt));
}
```

Listing 9: ver_info.cpp.

Código clase **imagenes.cpp**:

```

Mat ver_informacion(int nfoto)
{
    return foto[nfoto].img;
}
```

Listing 10: imagenes.cpp.

Con esto ya tendríamos hecho la implementación de ver información de una imagen.

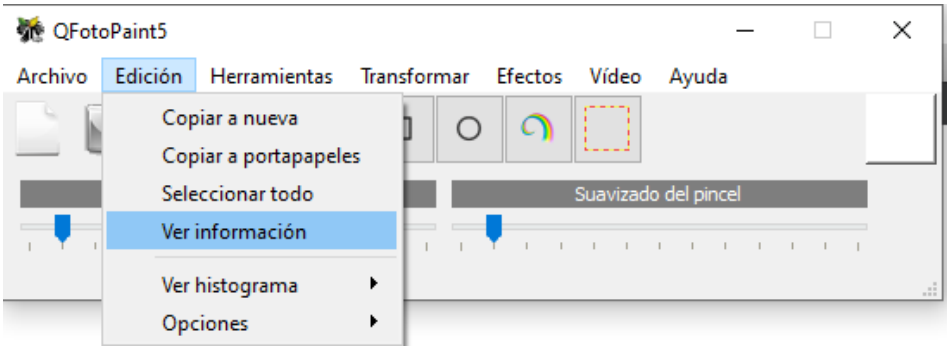

Figura 7: Menú desplegable para ver información.

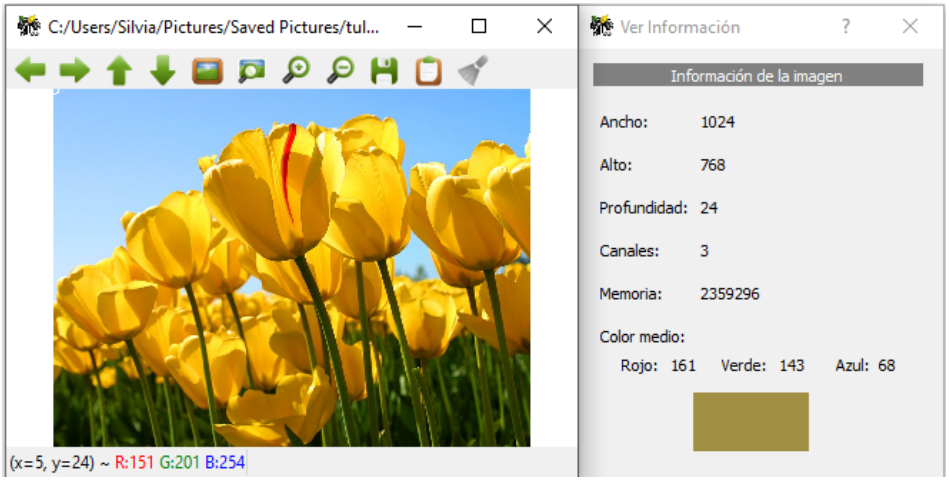

Figura 8: Ver información de la imagen.

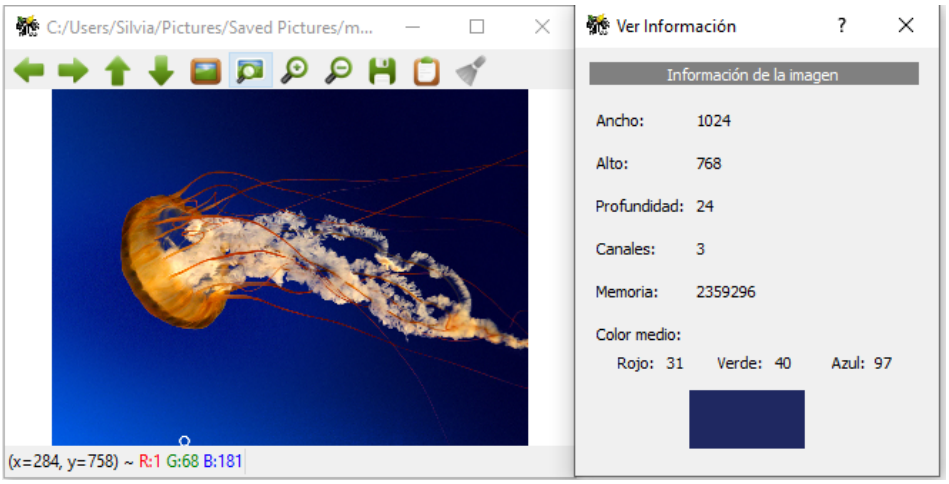

Figura 9: Ver información de la imagen.

4 Bloquear pincel

Esta propuesta se implementó porque en algunos casos nos dejaba usar el pincel encima de la imagen cuando se esta aplicando alguna trasformación. Para poder hacer que esto dejará de pasar es poniendo dos instrucciones, una en el constructor y otra en el desctructor. El del constructor es para poder bloquear el pincel mientras este activa la trasformación y cuando se destruye la ventana de la trasformación se vuelve a dejar el pincel sin bloquear. Esto esta implementado en las siguientes clases: ajustelineal.cpp, bajorrelieve.cpp, brillocontraste.cpp, convolucion.cpp, matt_sat_lum.cpp, pinchar_estirar.cpp, suavizados.cpp, color_falso.cpp, modelo_color.cpp y morfo_mat.cpp.

Por poner un código de ejemplo, vamos a mostrarlo con convolucion.cpp.

```
1
2  convolucion::convolucion(int numfoto, int numres, QWidget *parent) :
3      QDialog(parent),
4      ui(new Ui::convolucion)
5  {
6      ui->setupUi(this);
7      nfoto = numfoto;
8      //Ponemos a false el callback de esa foto
9      set_callback_foto(nfoto, false);
10     nres = numres;
11     M.create(5, 5, CV_64FC1);
12     actualizar();
13 }
14
15 convolucion::~convolucion()
16 {
17     // Lo ponemos a true.
18     set_callback_foto(nfoto, true);
19     delete ui;
20 }
```

Listing 11: convolucion.cpp.

5 Nueva desde el portapapeles

Para este opcional no hacia falta crear una nueva clase, para poder añadirlo lo único que hemos tenido que hacer es irnos al fichero **mainwindow.ui**, y en la pestaña de `Archivo` añadimos un nuevo campo que se llama `Abrir del portapapeles`.

Una vez hecho eso, tenemos que irnos al **mainwindow.cpp** e implementar la función triggered para poder llamar la implementación del opcional que se encuentra en **imagenes.cpp**. La función triggered se queda así:

```
1
2  void MainWindow::on_actionAbrir_del_porpapeles_triggered()
3  {
4      if (primera_libre() != -1)
5          pegar_del_portapapeles(primera_libre());
6  }
```

Listing 12: Función triggered().

La siguiente función que nos queda es la que consigue abrir la imagen del portapapeles. Primero recuperamos la imagen del clipboard general, si esa imagen es nula entonces mostramos un mensaje informando de que no hay ninguna imagen en el portapapeles. Si no es nula la imagen entonces la convertimos en un Mat y rellenamos con la información de la imq, cambiamos el formato de color y por último creamos la nueva imagen con el numero de imagen que nos pasen por parámetro y con la imagen creada.

```
1
2  void pegar_del_portapapeles(int nres)
3  {
4      QClipboard *clipboard = QGuiApplication::clipboard();
5      QImage imq = clipboard->image();
6
7      if (imq.isNull())
8          QMessageBox::information(w, "Informacion", "No hay ninguna imagen copiada en el
       portapapeles");
9      else {
10         Mat img(imq.height(),imq.width(),CV_8UC4,imq.scanLine(0));
11         cvtColor(img, img, COLOR_RGBA2RGB);
12         crear_nueva(nres, img);
13     }
14
15 }
```

Listing 13: Función pegar_del_portapapeles.

Con esto ya tendríamos hecho la implementación de pegar del portapapeles.

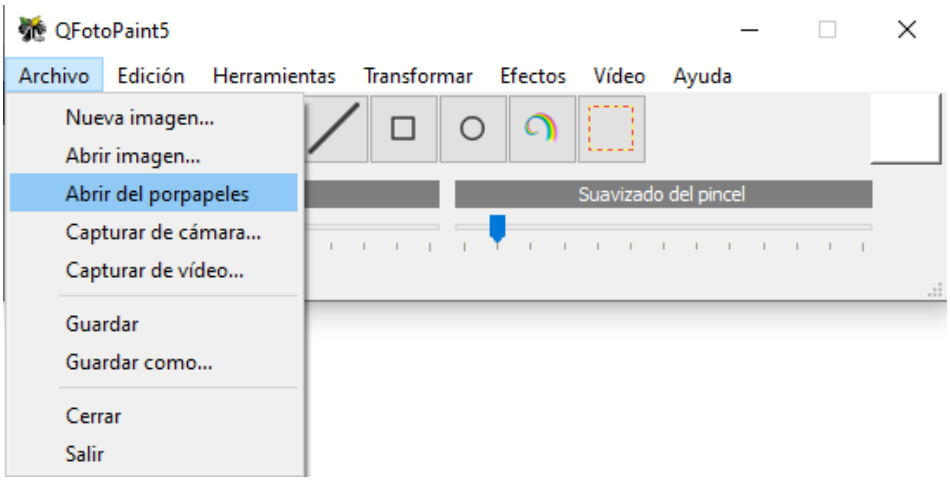

Figura 10: Menú desplegable para pegar del portapapeles.

Figura 11: Foto que previamente se había copiado en el portapapeles.

6 Convertir a color falso

Para este ejemplo hemos tenido que crear una clase llamada color_falso. Lo primero que hemos hecho ha sido implementar el diseño donde consta de cuatro radio buttons. Después nos hemos ido a al fichero **mainwindow.iu** donde hemos añadido la opción y hemos implementando el código para que lo conecte con el constructor de la clase **color_falso.cpp**. Por defecto dejamos la paleta de Twilight shifted activada. Dentro del constructor llamamos a la función donde se aplica el cambio de color. Esta función se encuentra en **imagenes.cpp** donde primero convertimos la imagen a una escala de grises y una vez que ya la tenemos, aplicamos la paleta con la función applyColorMap. Destacar que cada vez que se clicka en un checkbox se llama otra vez a la función pero con la paleta indicada.

Código en mainwindow.cpp:

```
1
2  void MainWindow::on_actionConvertir_a_color_falso_triggered()
3  {
4      if (foto_activa() != -1 && primera_libre() != -1) {
5          color_falso cf(foto_activa(), primera_libre());
6          cf.exec();
7      }
8  }
```

Listing 14: mainwindow.cpp.

Código en imagenes.cpp:

```
1
2  void convertir_a_color_falso(int nfoto, int nueva, int modo, bool guardar)
3  {
4      ColormapTypes paletas[4] = {COLORMAP_RAINBOW, COLORMAP_COOL, COLORMAP_PLASMA,
       COLORMAP_TWILIGHT_SHIFTED};
5
6
7      Mat gris;
8      cvtColor(foto[nfoto].img, gris, COLOR_BGR2GRAY);
9
10     Mat imgC;
11     applyColorMap(gris, imgC, paletas[modo]);
12
13     if (guardar)
14         crear_nueva(nueva, imgC);
15     else
16         imshow("Imagen Color Falso", imgC);
17 }
```

Listing 15: imagenes.cpp.

Código en color_falso.h:

```
1
2  #ifndef COLOR_FALSO_H
3  #define COLOR_FALSO_H
4
5  #include <QDialog>
6
7  namespace Ui {
8  class color_falso;
9  }
10
11 class color_falso : public QDialog
12 {
13     Q_OBJECT
14
15 public:
```

```
16      explicit color_falso(int num_foto, int nueva, QWidget *parent = nullptr);
17      ~color_falso();
18
19  private slots:
20      void on_radioButton_clicked();
21
22      void on_radioButton_2_clicked();
23
24      void on_radioButton_3_clicked();
25
26      void on_radioButton_4_clicked();
27
28      void on_color_falso_accepted();
29
30      void on_color_falso_rejected();
31
32  private:
33      Ui::color_falso *ui;
34      int paleta;
35      int nfoto;
36      int nres;
37  };
38
39  #endif // COLOR_FALSO_H
```

Listing 16: color_falso.h.

Código en color_falso.cpp:

```
1
2   #include "color_falso.h"
3   #include "ui_color_falso.h"
4   #include "imagenes.h"
5
6   color_falso::color_falso(int numfoto, int nueva, QWidget *parent) :
7       QDialog(parent),
8       ui(new Ui::color_falso)
9   {
10      ui->setupUi(this);
11      nfoto = numfoto;
12      set_callback_foto(nfoto, false);
13      nres = nueva;
14      paleta = 3;
15      convertir_a_color_falso(nfoto, nres, paleta);
16  }
17
18  color_falso::~color_falso()
19  {
20      delete ui;
21      set_callback_foto(nfoto, true);
22  }
23
24  void color_falso::on_radioButton_clicked()
25  {
26      paleta = 0;
27      convertir_a_color_falso(nfoto, nres, paleta);
28  }
29
30  void color_falso::on_radioButton_2_clicked()
31  {
32      paleta = 1;
33      convertir_a_color_falso(nfoto, nres, paleta);
34  }
35
36  void color_falso::on_radioButton_3_clicked()
37  {
38      paleta = 2;
39      convertir_a_color_falso(nfoto, nres, paleta);
40  }
41
42  void color_falso::on_radioButton_4_clicked()
43  {
44      paleta = 3;
```

```
45      convertir_a_color_falso(nfoto, nres, paleta);
46  }
47
48  void color_falso::on_color_falso_accepted()
49  {
50      convertir_a_color_falso(nfoto, nres, paleta, true);
51      destroyWindow("Imagen Color Falso");
52  }
53
54  void color_falso::on_color_falso_rejected()
55  {
56      destroyWindow("Imagen Color Falso");
57  }
```

Listing 17: color_falso.cpp.

Con esto ya tendríamos implementando la conversión a color falso.

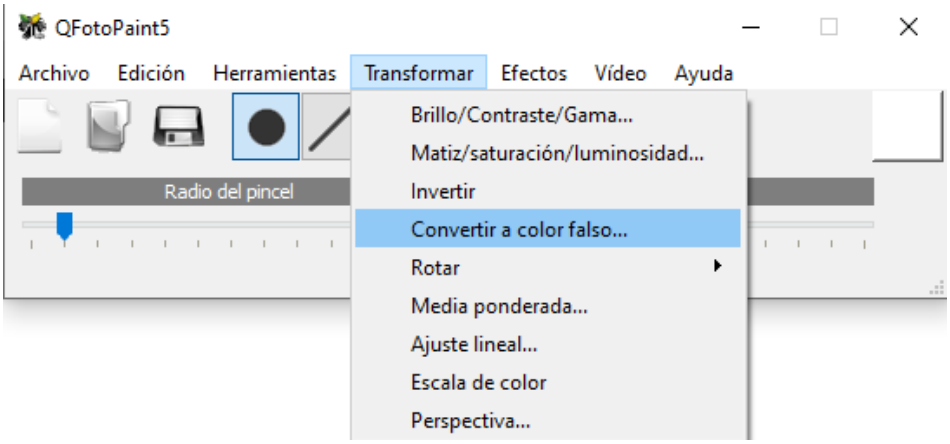

Figura 12: Menú desplegable para convertir a color falso.

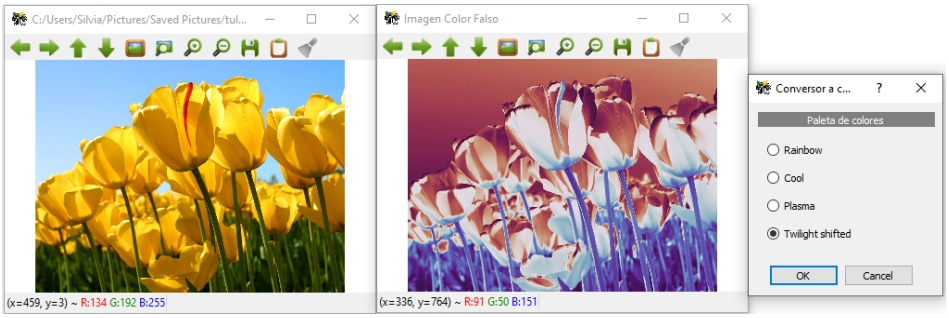

Figura 13: Ventana de color falso con twilight shifted.

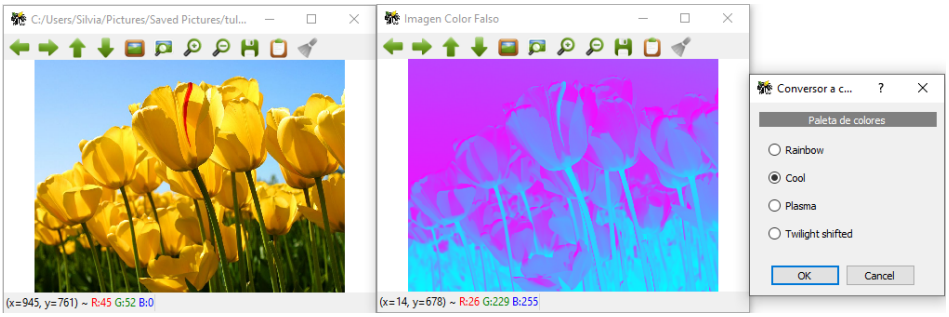

Figura 14: Ventana de color falso con cool.

7 Modelos de color

El proceso es parecido al anterior, creamos una nueva clase llamada modelo_color. Ponemos en los radio buttons los modelos a los que queremos convertir y aplicamos segun lo elegido. La función donde se hace el cambio simplemente tiene una instrucción donde cambia al color segun lo que pasen por parametro y lo hace con cvtColor.

```
1
2  void modelo_de_color(int nfoto, int nueva, int modelo, bool guardar)
3  {
4      int modelos[5] = {COLOR_BGR2RGB, COLOR_BGR2HLS, COLOR_BGR2HSV, COLOR_BGR2XYZ,
       COLOR_BGR2YUV};
5
6      Mat imgM;
7      cvtColor(foto[nfoto].img, imgM, modelos[modelo]);
8
9      if (guardar)
10         crear_nueva(nueva, imgM);
11     else
12         imshow("Nuevo modelo de color", imgM);
13 }
```

Listing 18: imagenes.cpp.

Código en mainwindow.cpp:

```
1
2  void MainWindow::on_actionModelos_de_color_triggered()
3  {
4      if (foto_activa() != -1 && primera_libre() != -1) {
5          modelo_color mc(foto_activa(), primera_libre());
6          mc.exec();
7      }
8  }
```

Listing 19: mainwindow.cpp.

Código en modelo_color.h:

```
1
2  #ifndef MODELO_COLOR_H
3  #define MODELO_COLOR_H
4
5  #include <QDialog>
6
7  namespace Ui {
8  class modelo_color;
9  }
10
11 class modelo_color : public QDialog
12 {
13     Q_OBJECT
14
15 public:
16     explicit modelo_color(int numfoto, int nueva, QWidget *parent = nullptr);
17     ~modelo_color();
18
19 private slots:
20     void on_radioButton_clicked();
21
22     void on_radioButton_2_clicked();
23
24     void on_radioButton_3_clicked();
25
26     void on_radioButton_4_clicked();
27
28     void on_radioButton_5_clicked();
29
```

```
30      void on_modelo_color_accepted();
31
32      void on_modelo_color_rejected();
33
34  private:
35      Ui::modelo_color *ui;
36      int nfoto;
37      int nres;
38      int modelo;
39  };
40
41  #endif // MODELO_COLOR_H
```

Listing 20: mainwindow.cpp.

Código en modelo_color.cpp:

```
1
2   #include "modelo_color.h"
3   #include "ui_modelo_color.h"
4   #include "imagenes.h"
5
6   modelo_color::modelo_color(int numfoto, int nueva, QWidget *parent) :
7       QDialog(parent),
8       ui(new Ui::modelo_color)
9   {
10      ui->setupUi(this);
11      nfoto = numfoto;
12      set_callback_foto(nfoto, false);
13      nres = nueva;
14      modelo = 0;
15      modelo_de_color(nfoto, nres, modelo);
16  }
17
18  modelo_color::~modelo_color()
19  {
20      set_callback_foto(nfoto, true);
21      delete ui;
22  }
23
24  void modelo_color::on_radioButton_clicked()
25  {
26      modelo = 0;
27      modelo_de_color(nfoto, nres, modelo);
28  }
29
30  void modelo_color::on_radioButton_2_clicked()
31  {
32      modelo = 1;
33      modelo_de_color(nfoto, nres, modelo);
34  }
35
36  void modelo_color::on_radioButton_3_clicked()
37  {
38      modelo = 2;
39      modelo_de_color(nfoto, nres, modelo);
40  }
41
42  void modelo_color::on_radioButton_4_clicked()
43  {
44      modelo = 3;
45      modelo_de_color(nfoto, nres, modelo);
46  }
47
48  void modelo_color::on_radioButton_5_clicked()
49  {
50      modelo = 4;
51      modelo_de_color(nfoto, nres, modelo);
52  }
53
54  void modelo_color::on_modelo_color_accepted()
55  {
56       modelo_de_color(nfoto, nres, modelo, true);
```

```
57      destroyWindow("Nuevo modelo de color");
58  }
59
60  void modelo_color::on_modelo_color_rejected()
61  {
62      destroyWindow("Nuevo modelo de color");
63  }
```

Listing 21: mainwindow.cpp.

Y con esto ya tendríamos implementado las transformaciones a diferentes modelos de color.

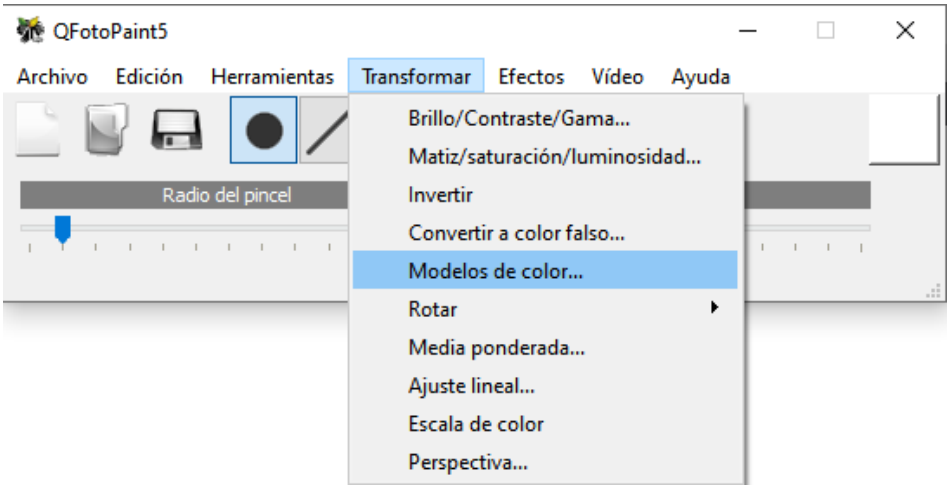

Figura 15: Menú desplegable para cambiar de modelo de color.

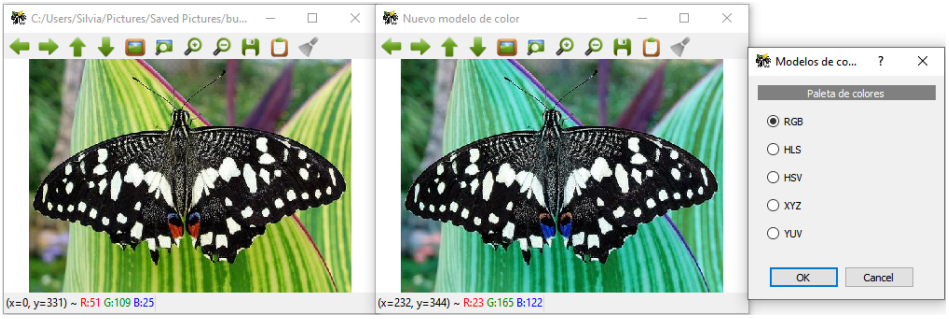

Figura 16: Ventana de cambio de color a RGB.

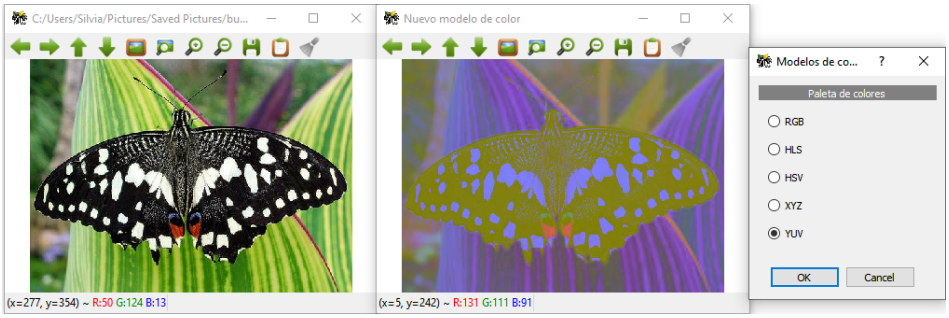

Figura 17: Ventana de cambio de color a YUV.

8 Morfología matemática

Se ha llevado a cabo la creación de un fichero nuevo llamado **morfo_mat.cpp** con su correspondiente **morfo_mat.h** y **morfo_mat.ui**. Además, hemos añadido al fichero **imagenes.cpp** e **imagenes.h** la función **morfologia_matematica** para poder aplicar las diferentes operaciones de morfología matemática.

En la función podemos ver que tenemos como argumentos de entrada aquellos parámetros para tratar la foto actual y donde mostrar la foto resultante, junto a los parámetros como el número de iteraciones y el modo para hacer la correspondiente operación de morfología matemática junto a un booleano para saber si queremos guardar la imagen.

Con el modo 0 se realiza la operación **dilatación**, con el modo 1 se realiza **erosión**, y por ultimo, las operaciones de **abrir** y **cerrar**

```
1
2  void morfologia_matematica(int nfoto, int nres, int nit, int modo, bool guardar)
3  {
4      Mat img = foto[nfoto].img;
5
6      Mat kernel = getStructuringElement(MORPH_RECT, cv::Size(3, 3));
7
8      Mat morfo_mat;
9
10     if (modo == 0)
11         dilate(img, morfo_mat, kernel, Point(-1, -1), nit);
12     else if (modo == 1)
13         erode(img, morfo_mat, kernel, Point(-1, -1), nit);
14     else if (modo == 2)
15         morphologyEx(img, morfo_mat, MORPH_OPEN, kernel, Point(-1, -1), nit);
16     else
17         morphologyEx(img, morfo_mat, MORPH_CLOSE, kernel, Point(-1, -1), nit);
18
19     if (guardar)
20         crear_nueva(nres, morfo_mat);
21     else
22         imshow("Morfolog a matem tica", morfo_mat);
23
24 }
```

Listing 22: morfologia_matematica.

A continuación mostramos los ficheros **morfo_mat.cpp** y **morfo_mat.h**. Mostramos que en el constructor de morfo_mat le añadimos los parámetros para obtener la foto actual y donde se mostrará. Como atributos añadimos el número de iteraciones y el modo, ambos inicializados a 0.

Disponemos de un spinBox y un horizontalSlider para que el usuario pueda seleccionar a su gusto el número de iteraciones, además de 4 radioButton para elegir la operación deseada.

```
1
2  #ifndef MORFO_MAT_H
3  #define MORFO_MAT_H
4
5  #include <QDialog>
6
7  namespace Ui {
8  class morfo_mat;
9  }
10
11 class morfo_mat : public QDialog
12 {
```

```
13      Q_OBJECT
14
15  public:
16      explicit morfo_mat(int numfoto, int nueva, QWidget *parent = nullptr);
17      ~morfo_mat();
18
19  private slots:
20      void on_radioButton_clicked();
21
22      void on_radioButton_2_clicked();
23
24      void on_radioButton_3_clicked();
25
26      void on_radioButton_4_clicked();
27
28      void on_horizontalSlider_valueChanged(int value);
29
30      void on_spinBox_valueChanged(int arg1);
31
32      void on_morfo_mat_accepted();
33
34      void on_morfo_mat_rejected();
35
36  private:
37      Ui::morfo_mat *ui;
38      int nfoto;
39      int nres;
40      int nit;
41      int modo;
42  };
43
44  #endif // MORFO_MAT_H
```

Listing 23: morfo_mar.h.

```
1
2   #include "morfo_mat.h"
3   #include "ui_morfo_mat.h"
4   #include "imagenes.h"
5
6   morfo_mat::morfo_mat(int numfoto, int nueva, QWidget *parent) :
7       QDialog(parent),
8       ui(new Ui::morfo_mat)
9   {
10      ui->setupUi(this);
11      nfoto = numfoto;
12      set_callback_foto(nfoto, false);
13      nres = nueva;
14      nit = 0;
15      modo = 0;
16      morfologia_matematica(nfoto, nres, nit, modo);
17  }
18
19  morfo_mat::~morfo_mat()
20  {
21      set_callback_foto(nfoto, true);
22      delete ui;
23  }
24
25  void morfo_mat::on_radioButton_clicked()
26  {
27      modo = 0;
28      morfologia_matematica(nfoto, nres, nit, modo);
29  }
30
31  void morfo_mat::on_radioButton_2_clicked()
32  {
33      modo = 1;
34      morfologia_matematica(nfoto, nres, nit, modo);
35  }
36
37  void morfo_mat::on_radioButton_3_clicked()
38  {
39      modo = 2;
```

```
40      morfologia_matematica(nfoto, nres, nit, modo);
41  }
42
43  void morfo_mat::on_radioButton_4_clicked()
44  {
45      modo = 3;
46      morfologia_matematica(nfoto, nres, nit, modo);
47  }
48
49  void morfo_mat::on_horizontalSlider_valueChanged(int value)
50  {
51      nit = value;
52      ui->spinBox->setValue(value);
53      morfologia_matematica(nfoto, nres, nit, modo);
54  }
55
56  void morfo_mat::on_spinBox_valueChanged(int arg1)
57  {
58      nit = arg1;
59      ui->horizontalSlider->setValue(arg1);
60      morfologia_matematica(nfoto, nres, nit, modo);
61  }
62
63  void morfo_mat::on_morfo_mat_accepted()
64  {
65      morfologia_matematica(nfoto, nres, nit, modo, true);
66      destroyWindow("Morfolog a matem tica");
67  }
68
69  void morfo_mat::on_morfo_mat_rejected()
70  {
71      destroyWindow("Morfolog a matem tica");
72  }
```

Listing 24: morfo_mar.cpp.

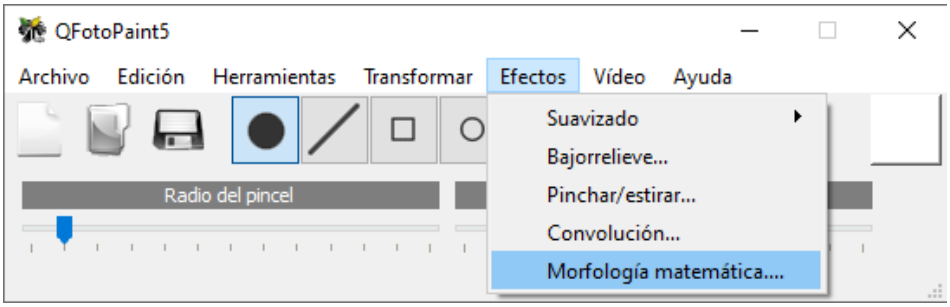

Figura 18: Menú desplegable de morfología matemática.

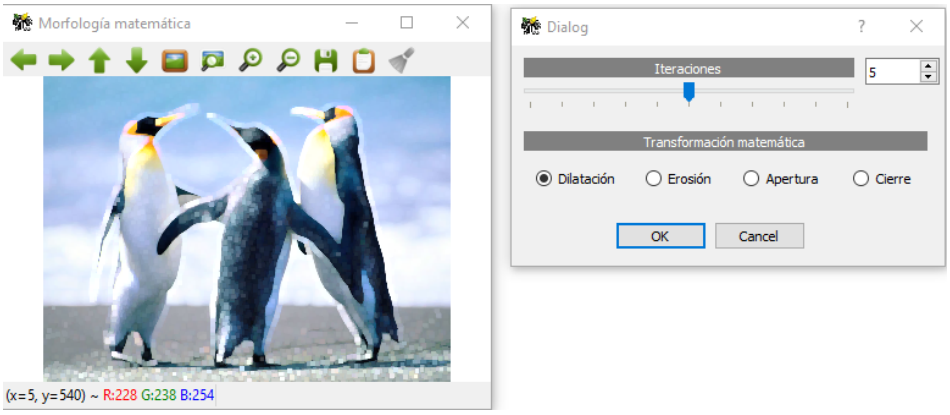

Figura 19: Operación de dilatación de morfología matemática.

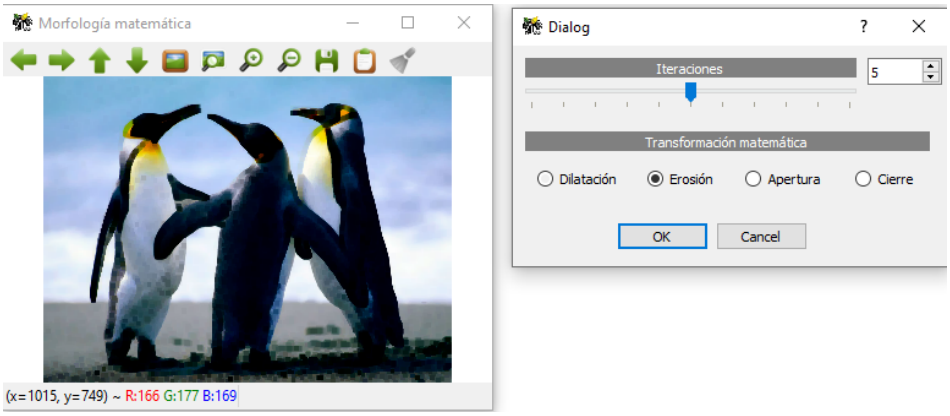

Figura 20: Operación de erosión de morfología matemática.

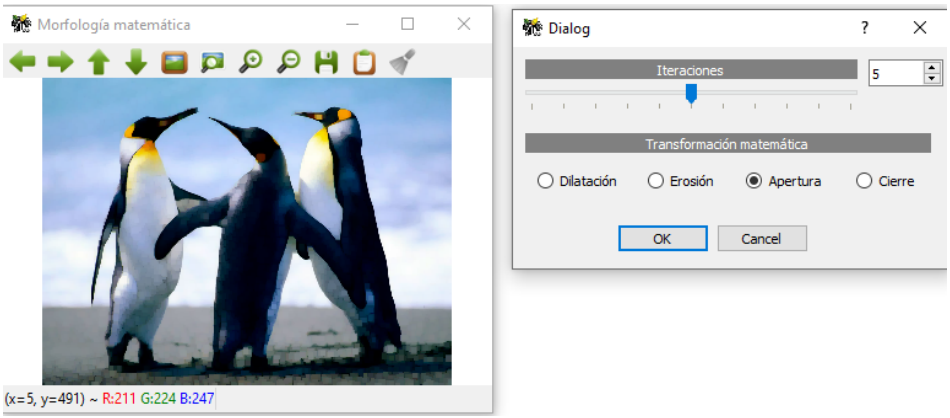

Figura 21: Operación de apertura de morfología matemática.

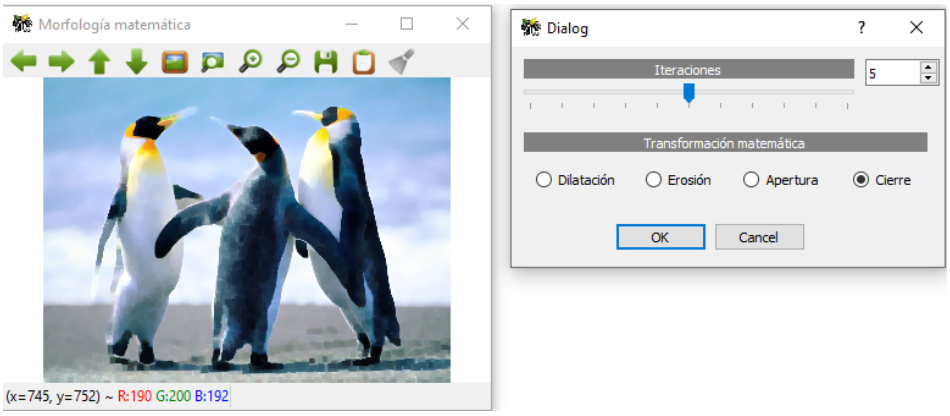

Figura 22: Operación de cierre de morfología matemática.