**Open CV Project**

**Group 1**

RAMÓN GONZÁLEZ, Miriam - 60577

RAMIRO BONILLA, Celia - 60576

Contents

[1. Tasks 2](#_Toc10577806)

[2. Functions 2](#_Toc10577807)

[Dilatation/Erosion 2](#_Toc10577808)

[Resizing 3](#_Toc10577809)

[Lighten/Darken 4](#_Toc10577810)

[Panorama/Stitching 5](#_Toc10577811)

[Canny Edge detection 6](#_Toc10577812)

[3. Additional functions 7](#_Toc10577813)

[Crop 7](#_Toc10577814)

[ColorMap 8](#_Toc10577815)

# 1. Tasks

|  |  |
| --- | --- |
| Functions | Student |
| Dilatation/Erosion |  |
| Resizing | Miriam RAMÓN GONZÁLEZ |
| Lighten/Darken |  |
| Panorama/Stitching |  |
| Canny Edge detection | Celia RAMIRO BONILLA |

|  |  |
| --- | --- |
| Additional Functions | Student |
| Crop | Miriam RAMÓN GONZÁLEZ |
| ColorMap | Celia RAMIRO BONILLA |
|  |  |

# 2. Functions

## Dilatation/Erosion

## Resizing

For this functionality is been created two functions:

1. void **Resize** (cv::Mat img);
   1. Store in a local variable the image that is passed as an argument.
   2. Create a window where the trackbar will be displayed.
   3. Create a trackbar to manage the factor of resizing.
      * The factor has a value between 0 and 600.
      * The trackbar starts in the mean value (300).
      * Initially two trackbars were created to manage the factor resizing of width and height. It works properly but it doesn’t keep the aspect ratio of the image.

Finally, only it is used a single trackbar to manage proportionally width and height in order to avoid image distortion.

* 1. Call to resizing function.

1. void **resizing** (int, void\*);
   1. Call to OpenCV resize function 🡪 void **resize** (ImputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation = *INTER\_LINEAR*);
      * src – It is the image stored in the first function provided as a parameter in the function main ().
      * dst – It is the resized image.
      * dsize – It is the output image size Size (factor +200, factor + 200)
      * fx –

fx, fy are the factor changed in the trackbar.

* + - fy –
    - Interpolation method:
      * **INTER\_NEAREST** - a nearest-neighbor interpolation
      * **INTER\_LINEAR** - a bilinear interpolation (used by default)
      * **INTER\_AREA** - resampling using pixel area relation. It may be a preferred method for image decimation, as it gives more - free results. But when the image is zoomed, it is similar to the INTER\_NEAREST method.
      * **INTER\_CUBIC** - a bicubic interpolation over 4x4 pixel neighborhood
      * **INTER\_LANCZOS4** - a Lanczos interpolation over 8x8 pixel neighborhood
  1. Create a window where the final image will be displayed.
  2. Display the final image.

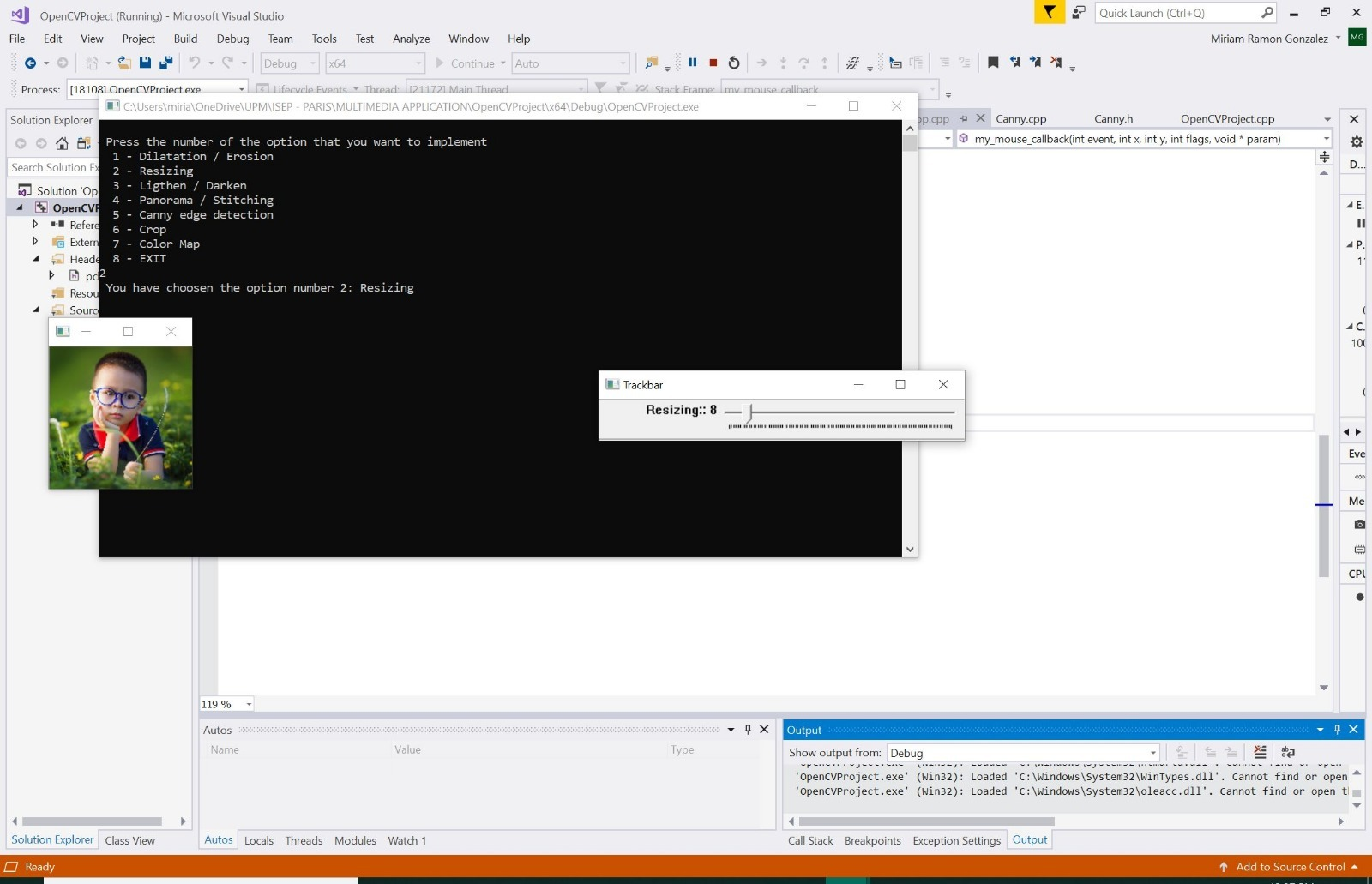


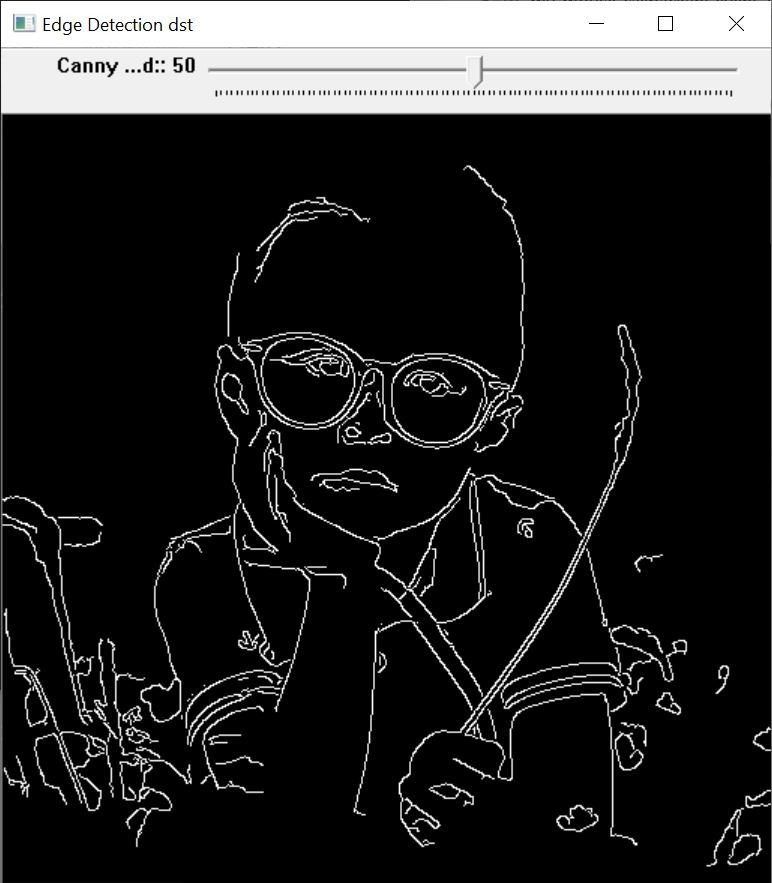
FIgure 1. Resizing a picture

## Lighten/Darken

## Panorama/Stitching

## Canny Edge detection

For this functionality is been created two functions:

1. void **Canny** (cv::Mat img);
   1. Store in a local variable the image that is passed as an argument by the main function.
   2. Create a matrix destination of the same type and size as src.
   3. Create a window where the trackbar will be displayed. This window will be the same for the final image.
   4. Create a trackbar to manage the factor of canny edge detection.
      * The factor has a value between 0 and 100.
      * The trackbar starts in the mean value (50).
   5. Call to resizing function.
2. void **edgeDetection** (int, void\*);
   1. The noise is reduced with a kernel 3x3 with function *blur*.

* Blurs an image using the normalized box filter,
* void **blur** (**InputArray src, OutputArray dst, Size ksize, Point anchor**=Point (-1, -1), **int borderType**=*BORDER\_DEFAULT*);
* **src** – input image.
* **dst** – output image of the same size and type as src.
* **ksize** – blurring kernel size.
* **anchor** – anchor point; default value Point (-1, -1) means that the anchor is at the kernel center.
* **borderType** – border mode used to extrapolate pixels outside of the image.
  1. Call to OpenCV Canny function 🡪 void **Canny** (InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize = 3, bool L2gradient = false);
* **image** – input image.
* **edges** – output edge map; it has the same size and type as image.
* **threshold1**

First and second threshold for the hysteresis procedure, they are modified by the trackbar

* **threshold2**
* **apertureSize** – aperture size for the [Sobel()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#void%20Sobel(InputArray%20src,%20OutputArray%20dst,%20int%20ddepth,%20int%20dx,%20int%20dy,%20int%20ksize,%20double%20scale,%20double%20delta,%20int%20borderType)) operator.
* **L2gradient** – a flag, indicating whether a more accurate  L_2 norm  =\sqrt{(dI/dx)^2 + (dI/dy)^2}  should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default L_1 norm=|dI/dx|+|dI/dy|  is enough (L2gradient = false).
  1. Create a window where the final image will be displayed.
  2. Display the final image.

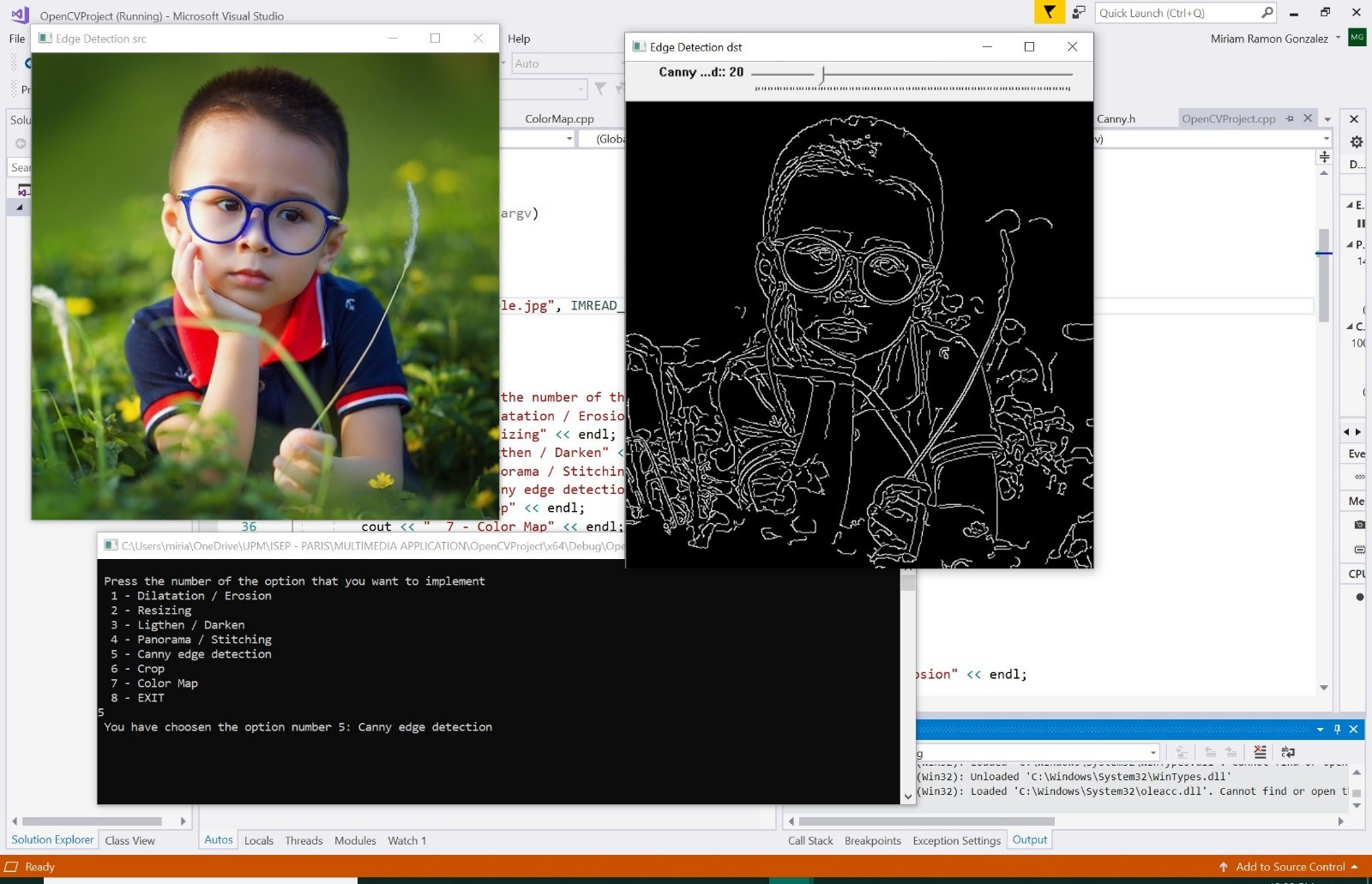


Figure 2. Canny

# 3. Additional functions

## Crop

The idea is cropping an image using the mouse according to the rectangle or square drawn. For this functionality is created two different functions:

1. void **Crop** (cv::Mat img);
   1. Store in a local variable the image that is passed as an argument by the main function.
   2. Create a window in order to display the source image.
   3. When a mouse event occurred is detected with the OpenCV ***setMouseCallback*** and*my\_mouse\_callback* function is called.
      * void setMouseCallback (const string& **winname**, MouseCallback **onMouse**, void\* **userdata**=0 )

* **winname** – Window name where recognize the mouse event.
* **onMouse** – Mouse callback to the function to process the event.
* **userdata** – The optional parameter passed to the callback.

1. void **my\_mouse\_callback** (int event, int x, int y, int flags, void\* param);
   1. Process the event detected with the OpenCV ***setMouseCallback*** in a switch case.
   2. In the switch case are processed:

When the left button down is detected, the movement of the mouse is processed and finally when the left button is up is detected the points of the shape drawn are saved as *roi (region of interest)* which is the final image.

* EVENT\_MOUSEMOVE
* EVENT\_LBUTTONDOWN
* EVENT\_LBUTTONUP
  1. Create a window where the final image will be displayed.
  2. Display the final image.

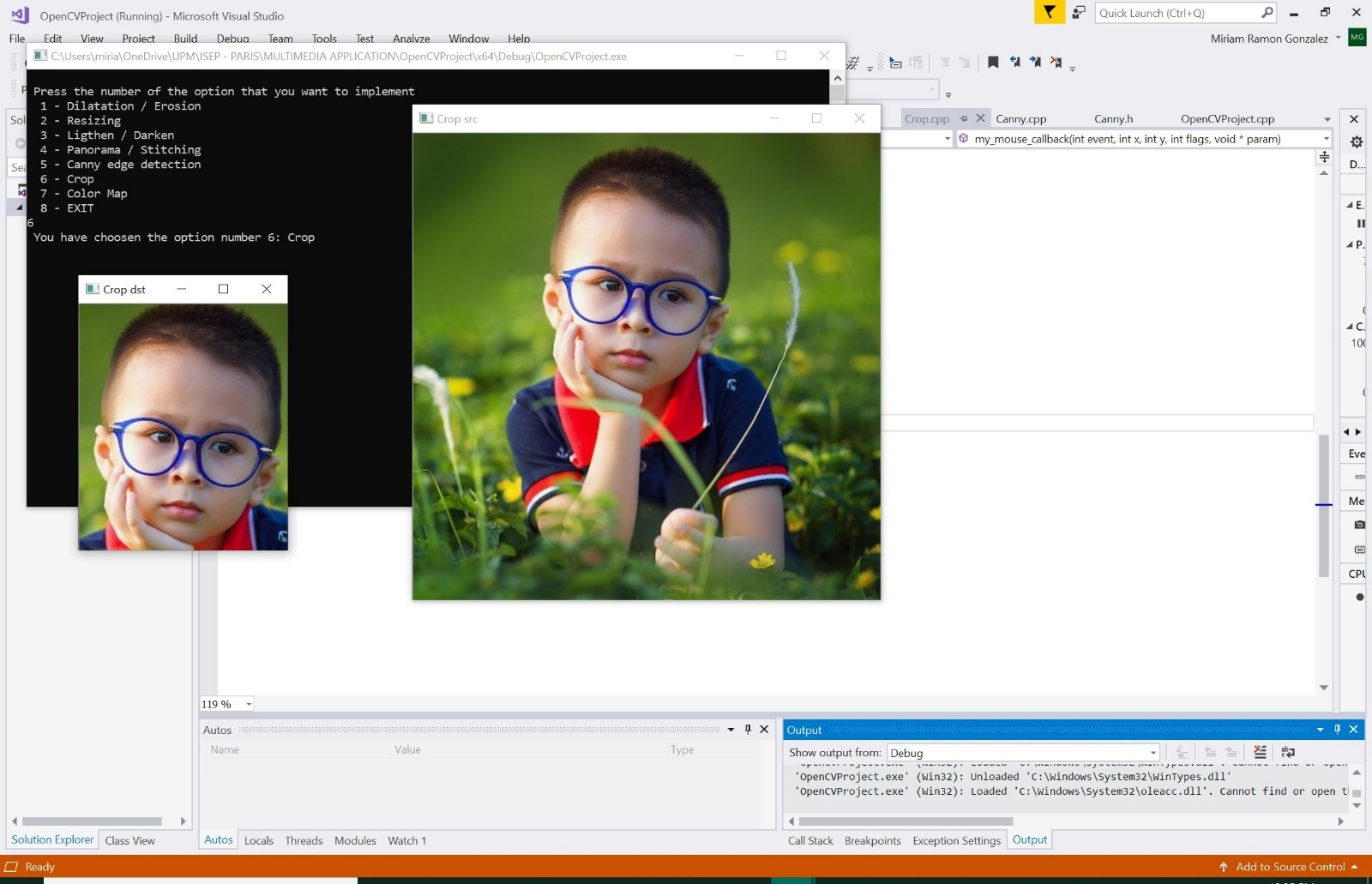
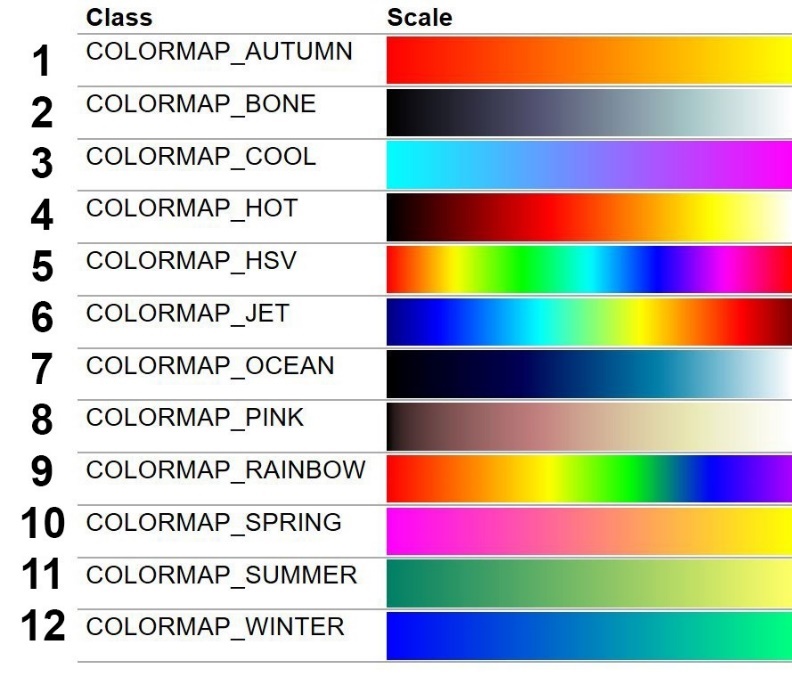


Figure 3. Cropping a picture

## ColorMap

The idea is creating some image filters or effects to change the colors like in an editor. For this functionality is created two different functions:

1. void **showInfo** ();
   1. Show the information about the filters with numbers as a main menu.
2. void **ColorMap** (cv::Mat img);
   1. Store in a local variable the image that is passed as an argument by the main function.
   2. Show the menu information calling **showInfo** ().
   3. Ask to the user for the number of the different filters to apply.
   4. If the number zero is chosen, this picture with extended information about the colors will be displayed.
   5. Process the number in a switch case
   6. The color is applied with the OpenCV **applyColorMap** function 🡪 void **applyColorMap** (InputArray src, OutputArray dst, int colormap);
      * **src** – input image.
      * **dst** – output image
      * **colormap** – filter applied
   7. Create a window where the final image will be displayed.
   8. Display the final image.

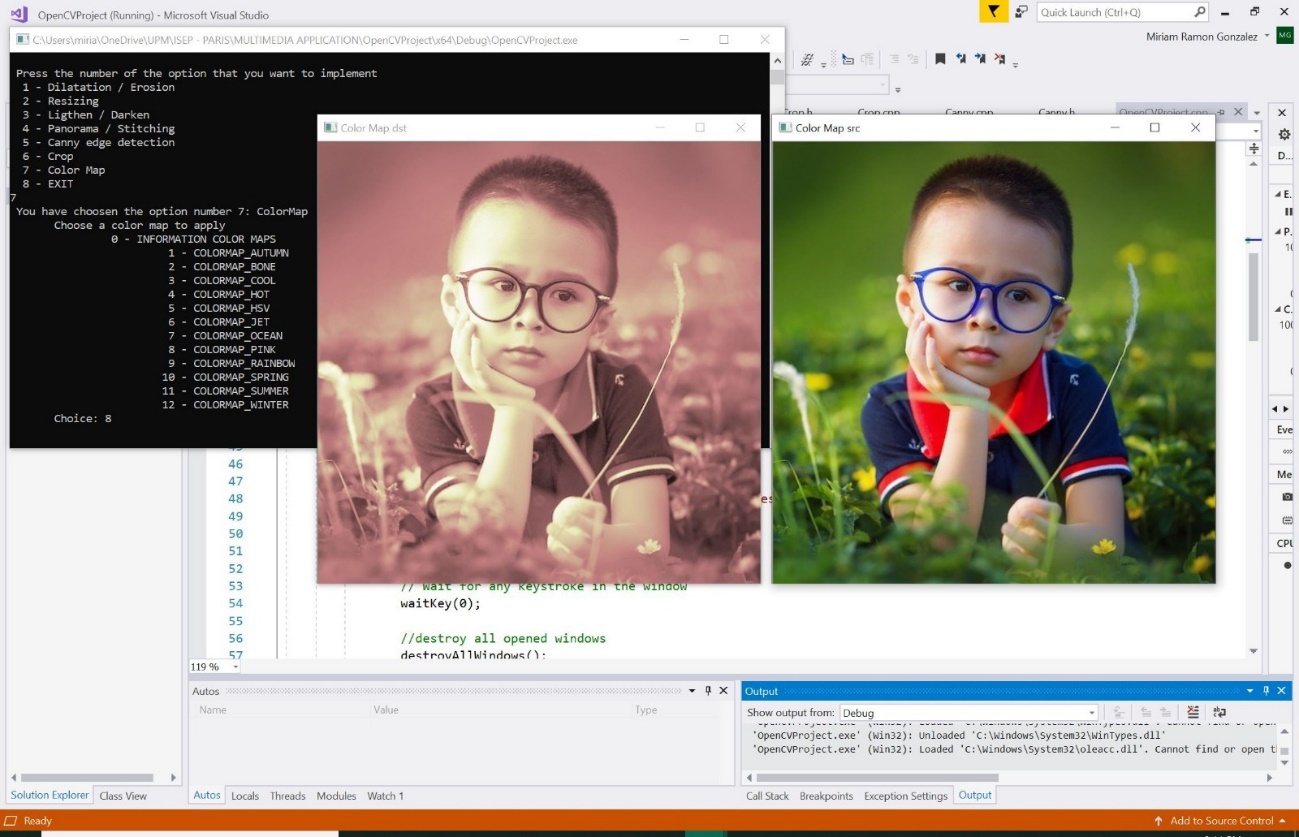


Figure 4. ColorMap with Pink filter.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

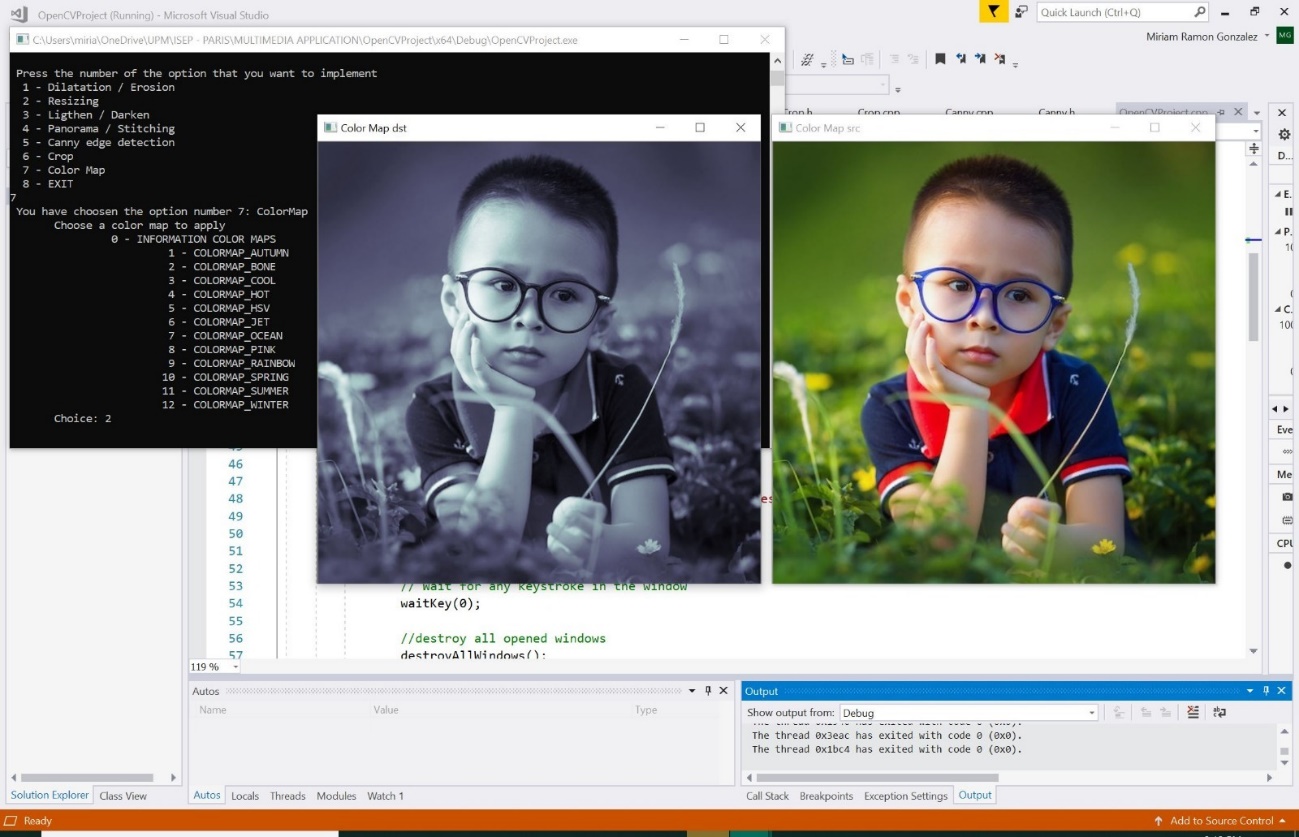


Figure 5. ColorMap with Bone filter.