Robotic Vision:

Assignment 3:
Geometric Transformations of images in ANSI C

José Ramón Romero Chávez (A01700318)
A01700318@itesm.mx

Tecnológico de Monterrey, Campus Querétaro

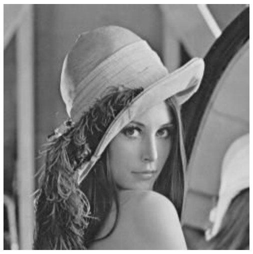
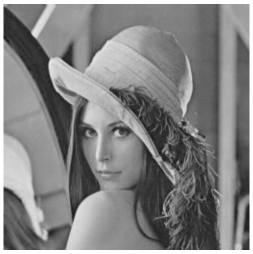
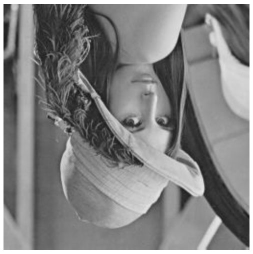
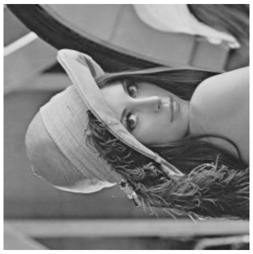
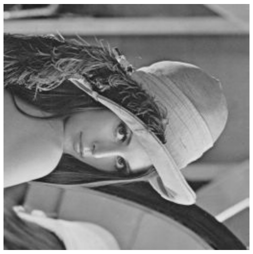
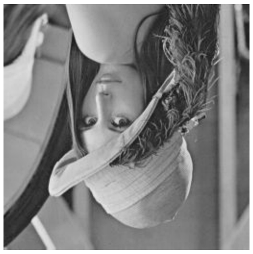
5th February 2019

Background:

To have a basic understanding of how images are handled and manipulated, ANSI C provides an excellent programming language. Geometric transformation is the manipulation of images in order to rotate, mirror, scale down, scale of, and flip them.

Objective:

Using LenaGrey 512x512.pgm image file we have been using in class and the template I gave you, write 5 different C programs that do the following image geometric transformations:

		
Original image	Flip horizontal	Flip vertical
		
Rotate left	Rotate right	Rotate 180
Image geometrics transformations to perform		

Upload the following:

- *Upload to the 5 programs written in C.*
- *Upload your 5 manipulated images as evidence.*
- *Include a report, written in word, which explains how your algorithm works for each case. This could be done using a flow chart, block diagram or any other representation.*

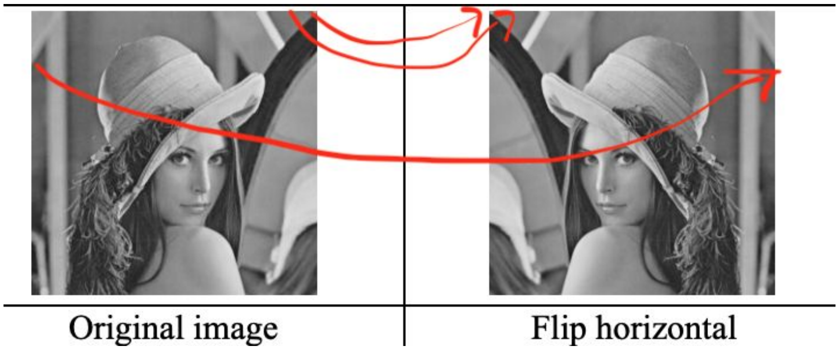
General Implementation

Inside of the `userdefined` procedure, it was necessary to read the original image, in order to get a more “readable” idea of the image and the operations to execute; it was stored on a allocated matrix that we are able to access by using the indexes of each of the char/bytes (aka “pointers of pointers”).

```
void userdefined ()
{
     unsigned char p;
     p=fgetc(infptr);
     int i,j;
     //Allocate a matrix
     unsigned  char **arr = (unsigned char **)malloc(MRows *
sizeof(unsigned char *));
    for (i=0; i<MRows; i++) {
     arr[i] = (unsigned char *)malloc(NCols * sizeof(unsigned char));
    }
    // Note that arr[i][j] is same as *(*(arr+i)+j), fill matrix
    for (i = 0; i <  MRows; i++){
      for (j = 0; j < NCols; j++){
       arr[i][j] = p;
       p=fgetc(infptr);
      }
    }

//---------------------------------------------------------------//
//                                                               //
//   … HERE GOES THE PARTICULAR IMPLEMENTATION FOR EACH CASE …   //
//                                                               //
//---------------------------------------------------------------//


}  // end userdefined ()
```

flipHorizontal

```
// write on output pointer flipHorizontal

    for (i = 0; i <  MRows; i++) {
      for (j = NCols-1; j >= 0; j--) {
         fputc(arr[i][j], outfptr);
         }
    }
```



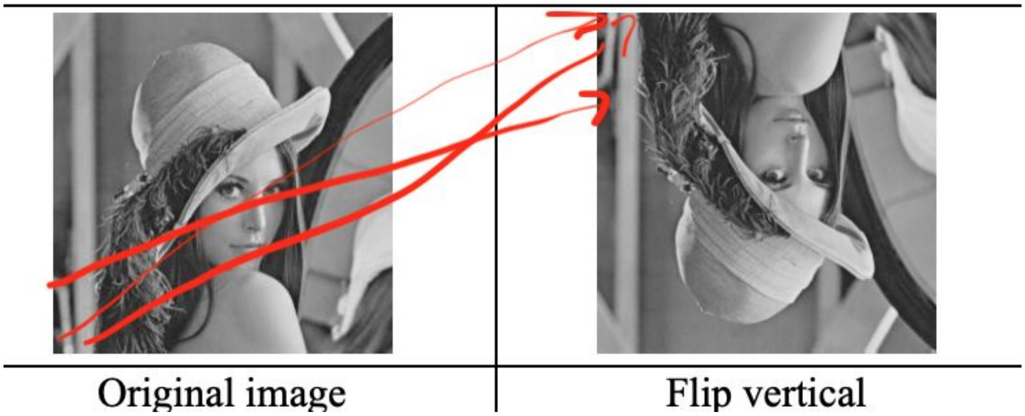
Original image

Flip horizontal

In this case is essential to show that the new images starts in reversed order in the horizontal axis but remains the same in vertical.

flipVertical

```
// write on output pointer flipVertical

    for (i = MRows-1; i >= 0; i--) {
      for (j = 0; j < NCols; j++) {
         fputc(arr[i][j], outfptr);
         }
    }
```



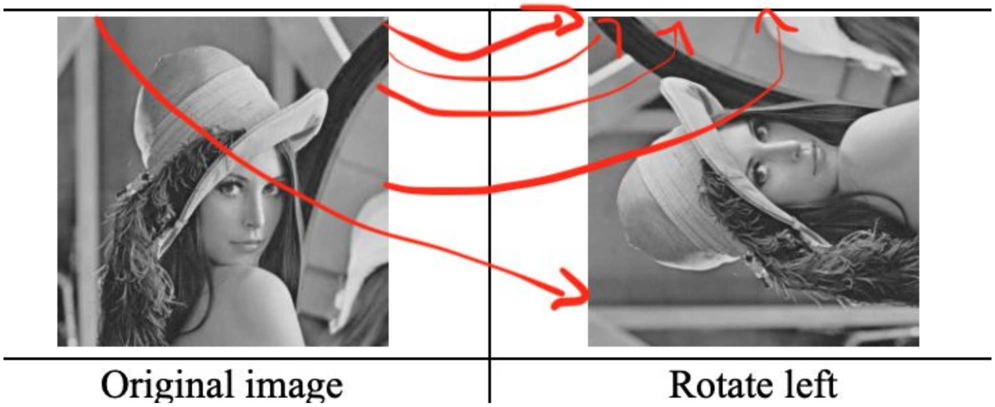
Original image

Flip vertical

In this case shown that the new images starts keeps order in the horizontal axis but it has reversed order in the vertical axis.

rotateLeft

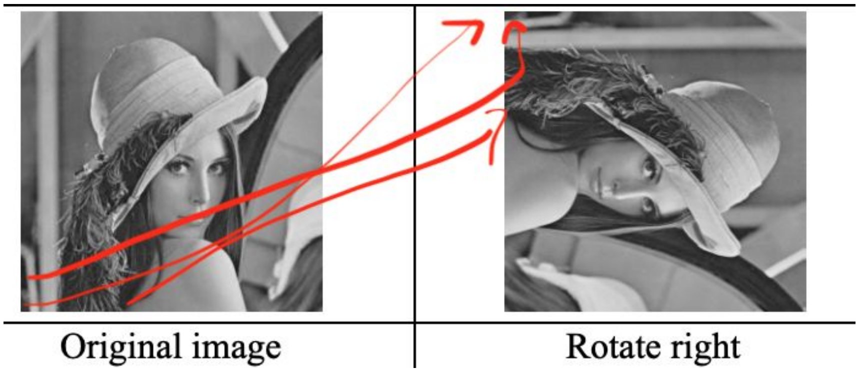
```
// write on output pointer rotateLeft

    for (i = NCols-1; i >= 0; i--) {
      for (j = 0; j < MRows; j++) {
         fputc(arr[j][i], outfptr);
         }
    }
```



In this case the columns became rows and vice versa, where our original and horizontal axis became a reversed column and the original-vertical only changes to horizontal.

rotateRight

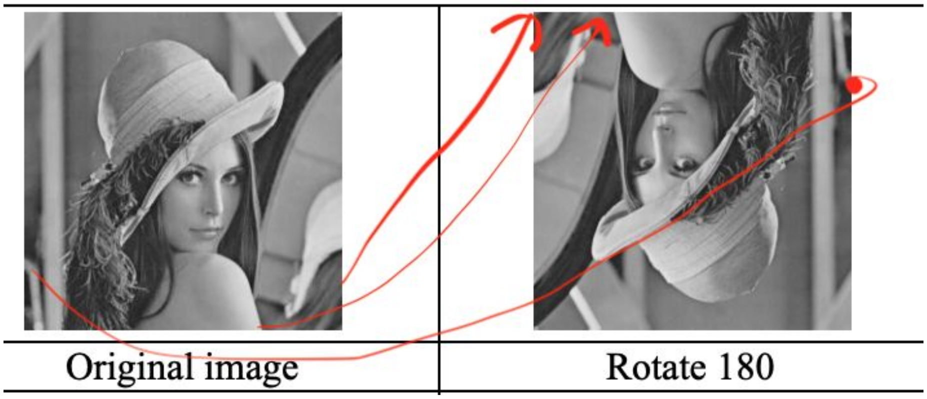
```
// write on output pointer rotateRight

    for (i = 0; i < NCols; i++) {
      for (j = MRows-1; j >=0 ; j--) {
         fputc(arr[j][i], outfptr);
         }
    }
```



In this case the columns became rows and vice versa, where our original and horizontal axis became a column and the original-vertical changes to a reversed horizontal.

rotate180

```
// write on output pointer rotate180

    for (i = MRows-1 ; i  >=0 ; i--) {
      for (j = NCols-1; j >=0 ; j--) {
         fputc(arr[i][j], outfptr);
         }
    }
```



Original image

Rotate 180

In this case the columns both axes, remain with the same orientation but is necessary to reverse the order inside them.