影像仿射轉換
(Using Visual studio C++ with Opencv)

何清模

January 16, 2024

Abstract

本作品是利用 Visual studio C++ 作為開發環境，並借助跨平台的電腦視覺函式庫 Opencv 的工具，將點陣圖型式的彩色照片讀取進來後，利用仿射變換的演算法，將影像放大、縮小、旋轉 (含 180 度) 調整，為減少調整後的失真情形，利用線性內插法或 median 法進行優化處理。

Contents

1 標頭檔及參考函式庫設定 **2**

2 Image processing **3**
2.1 Enlarge image by factor of 2 3
2.2 Shrink image by factor of 0.5 4
2.3 Rotation by degree of 15 4
2.4 Rotation by degree of 180 5

Chapter 1

標頭檔及參考函式庫設定

code:

```
#include <iostream>
#include "opencv2/opencv.hpp"
#include <opencv2/highgui/highgui.hpp>
#include<opencv2/core/utility.hpp>
#define pi 3.14159 // define the value of pi
using namespace std;
using namespace cv;
```

Chapter 2

Image processing

2.1 Enlarge image by factor of 2

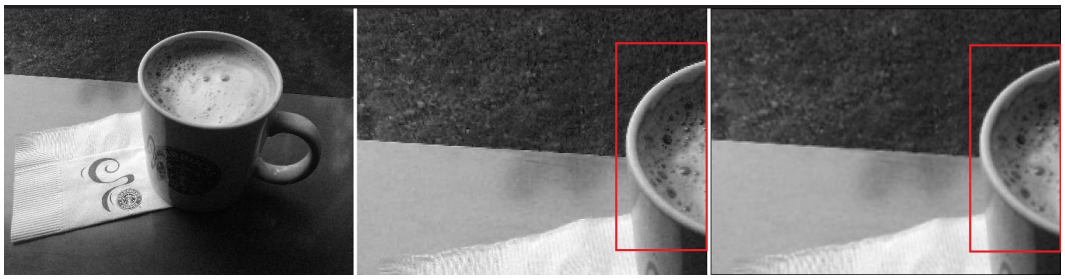

Figure 2.1: **Left subfigure:** original image; **Middle subfigure:** enlarged image without optimization; **Right subfigure:** enlarged image with optimization by using linear interpolation.

從 **Middle subfigure** 可以看出來，經過放大之後會產生失真現象，特別是在紅框處的杯緣處會有明顯的鋸齒狀；後經影像內插法的調整後 (如 **Right subfigure** 紅框處) 可發現，鋸齒狀的情形變得相對平滑許多。

2.2 Shrink image by factor of 0.5

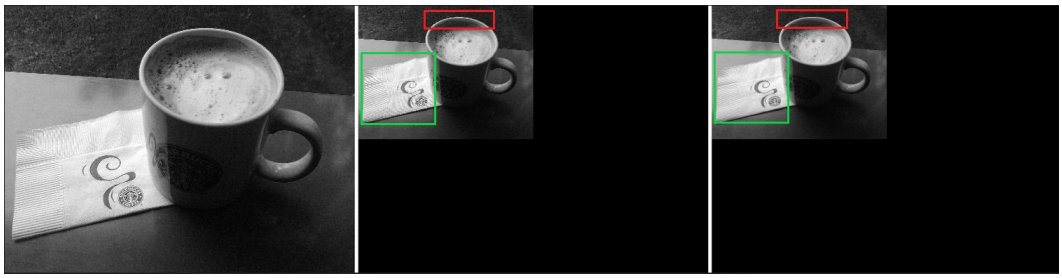

Figure 2.2: **Left subfigure:** original image; **Middle subfigure:** Shrinking image without optimization; **Right subfigure:** Shrinking image with optimization by using weighted value.

從 **Middle subfigure** 可以看出來，經過縮小之後會產生失真現象，如紅框處的杯緣鋸齒狀及綠框處的紙巾的陰影比起原圖顯得較不自然；後經影像像素權重法的調整後 (如 **Right subfigure** 紅框處及綠框處) 可發現，鋸齒狀的情形變得相對平滑許多，此外紙巾的陰影也與原圖較為相近。

2.3 Rotation by degree of 15

Figure 2.3: **Left subfigure:** original image; **Middle subfigure:** Rotating image without optimization; **Right subfigure:** Rotating image with optimization by using median filter.

從 **Middle subfigure** 可以看出來，經過座標轉換後，由於像素位置座標在轉換的過程當中，會從帶有小數型態的數值強迫變成整數型態，因此會有部分像素座標會無法一對一映射至轉換後的座標位置 (部分座標會重疊) 進而產生失真現象，如影像中類似椒鹽雜訊的孔洞情形 (因影像像素值初始化皆為 0)；後經 median filter 調整後 (如 **Right subfigure**) 可發現，整體影像的孔洞狀情形已修復。

2.4 Rotation by degree of 180

Figure 2.4: **Left subfigure:** original image;
Right subfigure: Rotating image.

由於像素位置座標在轉換的過程當中會運用到三角函數運算，因此影像在旋轉 90 度 180 度的整數倍數角度的時候，計算的結果剛好會是整數的數值，所以不會發生類似椒鹽雜訊的孔洞現象。

此外，本次旋轉 180 度的方法並非採用像素位置座標轉換的方式，而是使用以下觀念的方法完成：

$$\text{Original image}(x,y)\begin{cases} y \to \text{上至下} \\ x \to \text{左至右} \end{cases} \overset{\text{mapping}}{\Rightarrow} \text{Rotating image}(x,y)\begin{cases} y \to \text{下至上} \\ x \to \text{右至左} \end{cases}$$