PROJECT REPORT

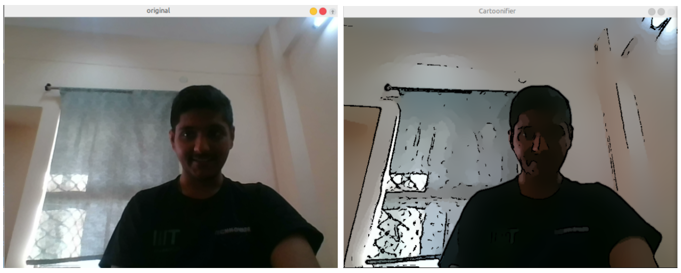
Cartoonification using Computer Vision
A Desktop App

Kushashwa Ravi Shrimali
December 31, 2017

1 INTRODUCTION

Computer Vision, the words define itself and it's importance. Making the computer understand the world around, is one of the biggest challenges in the present era. When you look around, you see many things happening. A camera captures moments that you cherish, though you might not notice this but every second, there is a moment that's happening around you. A still image to you, for a second, might not look that dynamic, but it is in the real world. No object has ever achieved absolute static state.

The aim of this project, lies in understanding the work of cartoonification that we see in mobile applications, and try and build one desktop application (a.k.a app) for the same.



(a) Original image as capture by camera. (b) Cartoonified Image as produced by application.

Figure 1.1: Outputs of the desktop app, left : original, right : output

The output and the input to the application has been provided in Figure 1.1. The application uses comprehensive use of filters, noise removal methods, edge detection filters, thresholding methods and more.

1.1 PROCESS

Let's start by blurring the image first using medianBlur function in OpenCV. It takes input of a grayscale image, that is the image has to have only one channel instead of 3. For this, first the image has to be converted to grayscale, and then blurred using medianBlur.

Parameters :

- src : input 1, 3, or 4 channel image when ksize is 3 or 5, the image depth should be CV_8U, CV_16U, or CV_32F, for larger aperture sizes, it can only be CV_8U.
- dst : destination array of the same size and type as src.
- ksize : aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

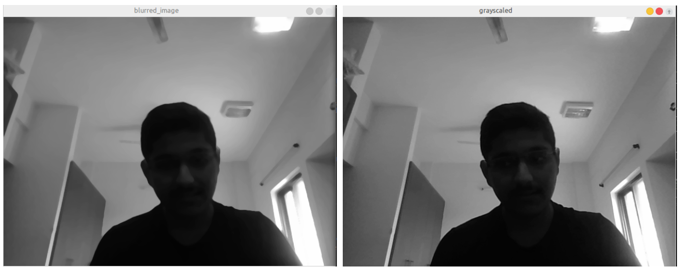
Resource: medianBlur

First task is to produce ***black and white sketch*** of the image. This has been done in two steps:

1. Edge Detection
2. Thresholding

In order to produce edge detected sketch, it's important to remove noise which can be done using medianBlur. Also, edge detection filters input grayscale image, so let's first convert the image to grayscale and apply medianBlur filter to reduce noise. Also, medianBlur filter is good at removing noise keeping sharp edges. Figure 1.2 shows the individual outputs from the following code.
Code for converting to grayscale and blurring using medianBlur filter :-

```
Mat gray;
// make gray image - blank
cvtColor(srcColor, gray, CV_BGR2GRAY);   // srcColor is the input image
const int MEDIAN_BLUR_FILTER_SIZE = 7; // kernel-size, odd number
medianBlur(gray, gray, MEDIAN_BLUR_FILTER_SIZE);
```



(a) Output after blurring using medianBlur filter [ksize=9]. (b) Image converted to grayscale.

Figure 1.2: Comparison of medianBlurring along with grayscaling.

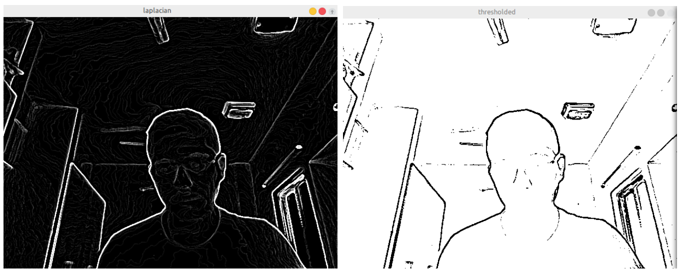
Once done, choice of edge detection filter had to be made. While Sobel and Scharr filters are comparatively slow, while Canny-edge filter produces unstable images as far as camera frames are concerned. So the best choice is Laplacian, which is comparatively fast as well as has stability with frames from camera.

Applying Laplacian filter to the blurred image:

```
const int LAPLACIAN_FILTER_SIZE = 5; // kernel size for Laplacian Filter
Laplacian(gray, edges, CV_8U, LAPLACIAN_FILTER_SIZE); // edges is Output image
```

After Laplacian Filter has been applied, it's time to threshold it to make it look like a sketch. Fig. 1.3 is how the output looks like.
Applying Thresholding to the filtered image:

```
const int EDGES_THRESHOLD = 80;
threshold(edges, masking, EDGES_THRESHOLD, 255, THRESH_BINARY_INV);
```



(a) Output image after Laplacian Filter been applied (b) Output image after thresholding.

Figure 1.3: Black and White sketch of original image.

Next task is to cartoonify the image, using Bilteral Filter. Applying bilateral filter individually to each frame from the camera, will take unacceptable time for any program. To speed up the process, bilateral filter has been applied at less resolution. For this, quarter of the pixels are taken from the image generated. Instead of large bilateral filters, many small bilateral filters are applied, for producing strong cartoon effect in less time.

Reducing resolution of the image (resizing)

```
Size smallSize;
smallSize.width = size.width/2;
smallSize.height = size.height/2;

Mat smallImg = Mat(smallSize, CV_8UC3);
resize(srcColor, smallImg, smallSize, 0, 0, INTER_LINEAR);
```

Once done, bilateral filters are applied of smaller kernel size (9). Bilateral filtering is an important tool in image processing, here is some important information about it:

```
void bilateralFilter(InputArray src, OutputArray dst, int d,
double sigmaColor, double sigmaSpace, int borderType=BORDER_DEFAULT )
```

Sigma values: For simplicity, you can set the 2 sigma values to be the same. If they are small (< 10), the filter will not have much effect, whereas if they are large (> 150), they will have a very strong effect, making the image look âĂIJcartoonishâĂİ.

Filter size: Large filters (d > 5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for offline applications that need heavy noise filtering.

Resource: Bilateral Filtering OpenCV

```
Mat tmp = Mat(smallSize, CV_8UC3);
int repetitions = 7;

for(int i = 0; i<repetitions; i++) {
  int ksize=9;
  double sigmaColor = 9;
  double sigmaSpace = 7;
  bilateralFilter(smallImg, tmp, ksize, sigmaColor, sigmaSpace);
  bilateralFilter(tmp, smallImg, ksize, sigmaColor, sigmaSpace);
}
```

Once all this has been done, it's time to overlap the filtered image with masked image such that the filtered image overlaps wherever there is no edge.

```
resize(smallImg, srcColor, size, 0, 0, INTER_LINEAR);
memset((char*)dst.data, 0, dst.step * dst.rows); // black background
srcColor.copyTo(dst, masking); // masking - masked image
```

1.2 References

1. Bilateral Filtering : Bilateral Filtering for Gray and Color Images

2. OpenCV Docs : [filters] Image Processing: Filters in OpenCV

3. Mastering OpenCV with Practical Computer Vision Projects, PACKT Publishing. Authors: Daniel Lelis Baggio, David Millan Escriva, Naureen Mahmood, Roy Shilkrot, Shervin Emami, Khvedchenia Levgen, Jason Saragih