CIS 4720 Image Processing and Computer Vision

Image Processing ToolBox

Noura Zekry

Table of Contents

Section I: User Manual

Overview:

My image processing toolbox allows its user to manipulate and apply different functions and transformations to a grayscale image of their choice. In addition, I have implemented a prototype of an object motion tracking program that I will describe in more detail.

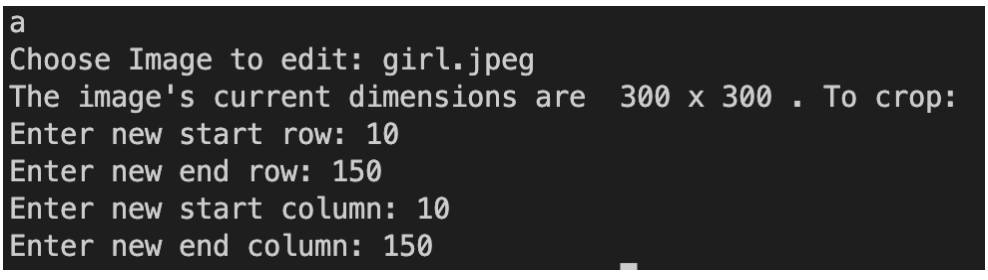
Audience:

The program assumes a user-base of experienced technologically-savvy users who are interested in interacting with a simple interface in order to explore and experiment with different image processing functions and transformations. The user interface is text-based and allows the user to interact with any of the 13 available functionalities.

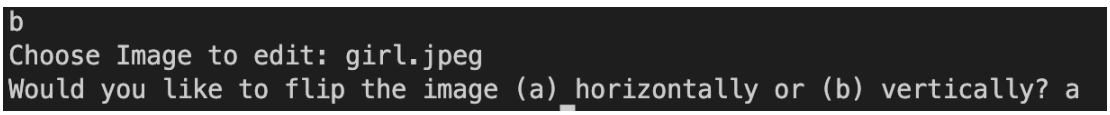
Instructions:

First, it prompts the user to choose a letter from a-m (lowercase) denoting a function they would like to explore. Options a-l will prompt the user to choose an image and enter its name and extension. Option m does not require an input image. The available options are:

a. Cropping:
    i. Upon choosing the crop option, the user will be informed of the image's current dimensions. In order to crop, the user must choose a start row, end row, start column, and end column. This means that if the goal were to crop an image of size 300x300px to new dimensions defined by the start point (10, 10) and end point (150, 150), the user would input 10 as the start row, 150 as the end row, 10 as the start column, and 150 as the end column.



b. Flipping an Image (reflection):
    i. The user will be presented with 2 options: a, to reflect horizontally, or b, to reflect vertically.



c. Scale:
    i. The user will be informed of the image's dimensions and then prompted to enter their choices of new vertical and horizontal dimensions. The user must input integer values greater than 1. Example of scaling a 300x300px image to 500x500px:

```
c
Choose Image to edit: girl.jpeg
The image's current dimensions are  300 x 300
Please enter new vertical dimension: 500
Please enter new horizontal dimension: 500
```

d. Rotate:
    i. The user is prompted to enter an angle of rotation which must be a real number greater than or equal to 0.
    ii.

```
d
Choose Image to edit: girl.jpeg
Please enter angle of rotation: 375
```

e. Linear Mapping:
    i. The user is asked to input parameter a, to denote gain or contrast, and then parameter b, to denote bias or brightness. The user should enter any real number in the form of a decimal or whole number.
    ii.

```
e
Choose Image to edit: girl.jpeg
Enter parameter a, contrast: 0.5
Enter parameter b, brightness: 2
```

f. Power-law Mapping:
    i. The user must enter a value **γ** to be applied in the power-law mapping $(L-1)[u/(L-1)]^{\gamma}$

```
f
Choose Image to edit: girl.jpeg
Enter gamma to be used for power-law mapping {(L-1)[u/(L -1)]^gamma}: 0.6
```

g. Histogram and Histogram Equalization:
    i. Once the user chooses this option, the program will calculate a histogram of the image of their choice and save it to "histogram.txt". They will be asked whether or not they would like to perform histogram equalization and transform the image to which they must respond with either a lowercase y to denote yes or lowercase n to denote no.
    ii.

```
g
Choose Image to edit: girl.jpeg
equalize? (y/n): y
```

h. Convolution:
    i. The user will be prompted to choose a CSV file from which to import a kernel of any integers MxN size. The given convolution will then be applied to their chosen image.

ii.
```
h
Choose Image to edit: girl.jpeg
CSV File to read kernel from: kernels/5GaussianKernel.csv
```

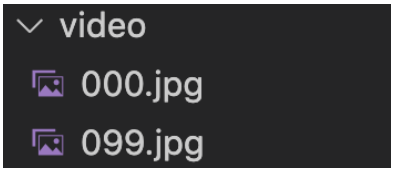
i. Edge Detection:
   i. The user is able to choose an image that will be processed through an edge detection algorithm using an inputted threshold (an integer from 0-255).
   ii.
```
i
Choose Image to edit: girl.jpeg
Enter edge detection threshold >= 0 and <= 255: 128
```

j. Min Filtering:
   i. This feature will apply min filtering to the image of choice. The user must input a positive integer size M to be used to apply a square min filtering kernel.
   ii.
```
j
Choose Image to edit: girl.jpeg
Enter size M for a kernel of size MxM: 5
```

k. Median Filtering:
   i. This feature will apply median filtering to the image of choice. The user must input a positive integer size M to be used to apply a square median filtering kernel.
   ii.
```
j
Choose Image to edit: girl.jpeg
Enter size M for a kernel of size MxM: 5
```

l. Max Filtering:
   i. This feature will apply max filtering to the image of choice. The user must input a positive integer size M to be used to apply a square max filtering kernel.
   ii.
```
l
Choose Image to edit: girl.jpeg
Input size M for kernel of size MxM: 10
```

m. Object Tracking Prototype:
   i. This feature does not require further user input and functions on the input images in the subdirectory “video”. It requires that there exist files titled 000.jpg and 099.jpg in said directory.



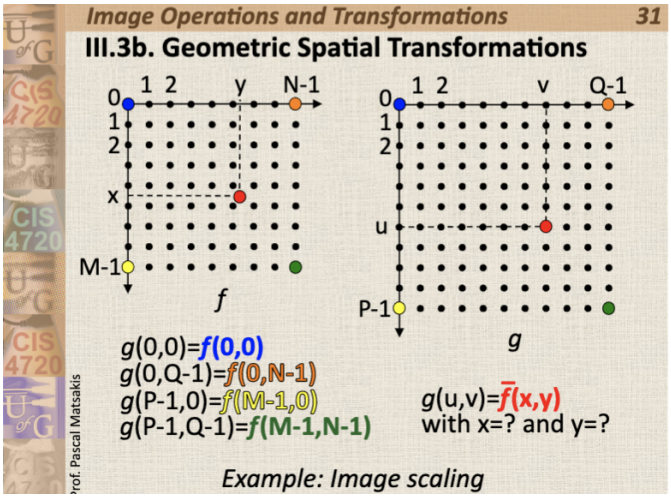
   ii. The output image can be found in the main directory under the name “ReverseSobel.jpeg”.

For all the functionalities available except for functionality “m” (Object Tracking), the output can be found in the file “edited.jpeg”

The program runs one functionality at a time. This means that if the user wishes to apply several transformations to the image they must run the program with the first transformation to their chosen image, retrieve the output image, and then run it again with the output image as the new input image.

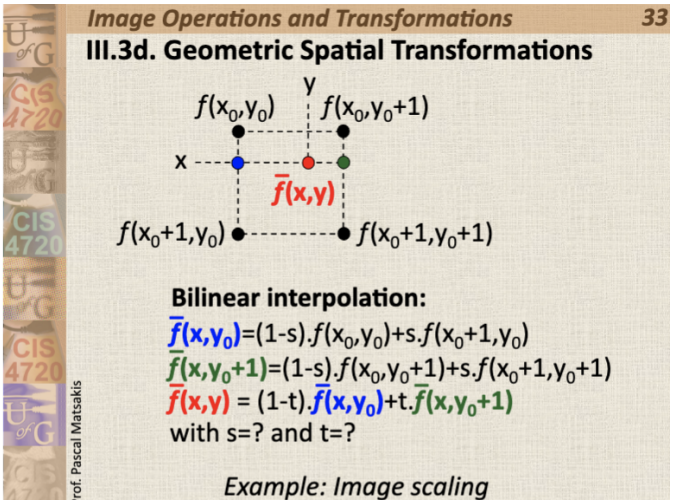
Section II: Technical Discussion

The program takes an input image of size MxN and transforms it into a numerical two-dimensional array of size MxN with values between 0-255. The ways in which the image array is processed and saved into a new image will be described below. Zero-padding is assumed where needed.

a. Crop:
    i. The underlying process behind the crop option is that it truncates the array using the provided indices. For instance, an image of size 300x300 px, and input of; start row: 10, end row: 150, start column: 10, end column: 150 will result in a truncated array of size 140x140. This array will then be saved as an image.
b. Reflection:
    i. Reflection happens through switching the values in the array over the given axis. A horizontal reflection results in a reflection over the x-axis while vertical reflection results in reflection over the y-axis.
c. Scale:
    i. Image scaling is the process of expanding or decreasing the horizontal and vertical ranges of the array. This requires a re-mapping of the values of the original image function to the new range values. Lecture Slides B p.31 part III.3b illustrate this process well.
    ii. 

1. In the above example, the original size is noted as MxN and the new size is PxQ. In order to complete the remapping, each x-value will be mapped as $x = u(M-1)/(P-1)$ and each y-value will become $y = v(N-1)/(Q-1)$

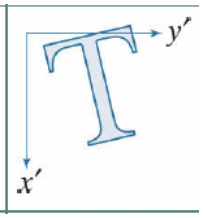
iii. Bilinear Interpolation is then performed to map the values into the discrete space Z^2. This is outlined in Lecture Slides B p.33 part III.3d:

1. 

d. Rotate:

i. Rotation involves a pixel-wise spatial transformation about an angle of choice. This requires a remapping of the origin point so as to maintain a viewable image and, thereby, may involve an expansion of the output image with zero padding. The pixel-wise transformation can be described as such:

ii.

Rotation (about the origin)	$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x\cos\theta - y\sin\theta$ $y' = x\sin\theta + y\cos\theta$	

e. Linear Mapping:

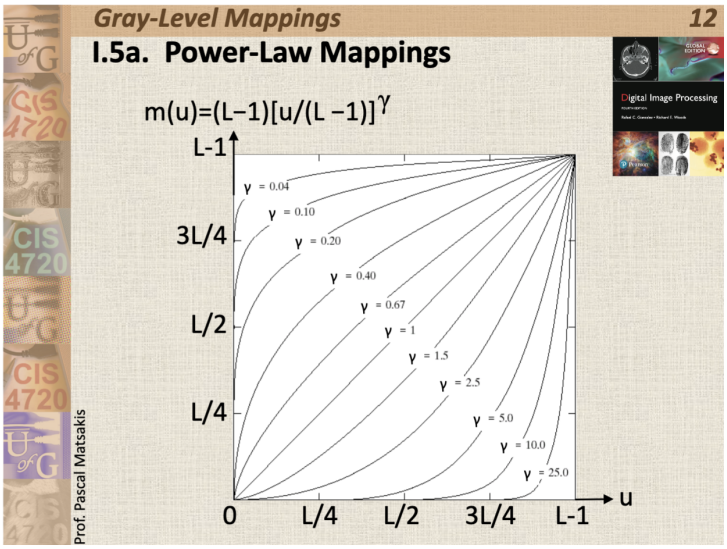
i. Linear mapping is the process of applying the function m(u) = au+b to every gray-level u in a two-dimensional image f where f(x,y) = u. This feature of the program allows the user to manipulate the parameters a and b to experiment with different levels of contrast and brightness, respectively.

f. Power-law Mapping:

i. Power-law mapping involves the application of the function $m(u)=(L-1)[u/(L-1)]^{\gamma}$ to every gray-level u in a two-dimensional image f where f(x,y) = u. This feature of the program allows the user to experiment with inputting different values for the parameter γ in order to tamper with the brightness and contrast of the image.

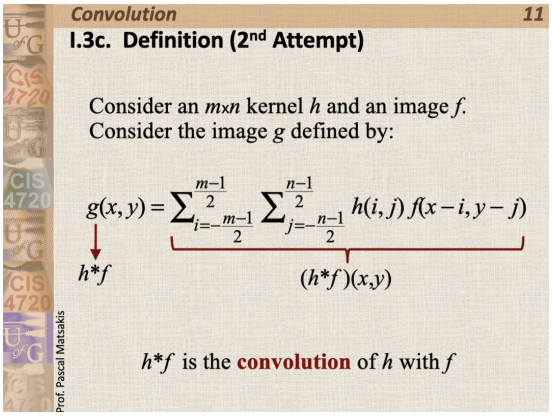
ii. The image below (Lecture Slides C p.12 part I.5b) provides a visualization of the different power-law mappings achievable through the manipulation of γ:

iii.



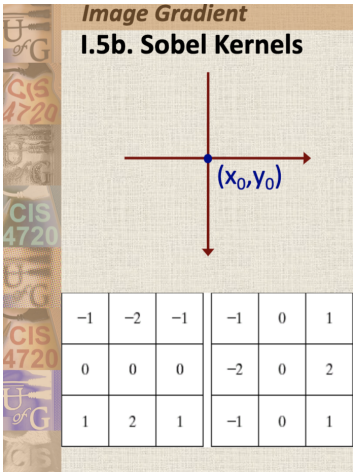
g. Histogram and Histogram Equalization:
   i. The histogram of an image represents the number of pixels of each gray-level from 0-255. This histogram can be equalized in order to approximate statistically optimal image enhancement. The equalization occurs by calculating the cumulative normalized histogram for an image f which can be defined as $H_f^{cn}(u)$ that describes the probability that a gray-level is less than or equal to u. The cumulative normalized histogram is then equalized by multiplying every value by the maximum gray-level (L-1) or 255 to convert it (with the help of the floor function) from a probability into an absolute gray-level. The image is then reproduced through flattening the image and mapping the values of the equalized histogram back to the image to be reproduced.
h. Convolution:
   i. Convolution is the process of manipulating an image f using a kernel h. The application of kernel h to kernel f can be described using the image below (Lecture Slides D p.11 1.3c.):

ii.



i. Edge Detection:
   i. Edge detection is implemented using Sobel convolution kernels. The program applies the vertical and horizontal convolution kernels. This detects the vertical and horizontal edges which are then used to calculate the gradient image along with the user's threshold input. The magnitude of the gradient image is used to create a binary edge map of the image to describe the edges. The sobel kernels used to detect the horizontal and vertical edges are described below using (Lecture Slides E p.13 part 1.5b.)

ii.



j. Min Filtering:
k. Median Filtering:
l. Max Filtering:
   i. Min, median, and max filtering function in essentially the same way with the exception of the choice of the order statistic property. For each of them a filter size is inputted. The filter size determines the neighborhood for each pixel from which either the min, median, or max will be chosen and assigned to the given pixel.
m. Object Tracking Prototype:
   i. This feature implements a prototype adapted from what is known as the "UnCanny Filter" or reversed Sobel edge detection developed by Honeycutt and Bridge (2021). The purpose of this filter is to detect the relevant motion between frames of a video

without detecting too much of the inevitable noise differences. My implementation works by deriving the binary edge maps (using the above describe Sobel Edge Detection), finding the difference between the two maps, then applying the UnCanny Filter which can be illustrated below:

$$\boldsymbol{M'} = \boldsymbol{M_n} - \boldsymbol{M_{n-1}}$$

$$\boldsymbol{M''} = \boldsymbol{G}\left(\boldsymbol{M'}_x, \boldsymbol{M'}_y\right) - \overline{\boldsymbol{G}\left(\boldsymbol{M'}_x, \boldsymbol{M'}_y\right)} + 255$$

ii.

1. M_n describes the current frame while M_{n-1} describes the previous frame. In my implementation, I chose to calculate the difference between the edges as that seemed to provide the best results.
2. G denotes the Gaussian blur. In my implementation, I employed the following 5x5 Gaussian kernel:

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

$$\boldsymbol{S}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \times \boldsymbol{M''}, \quad \boldsymbol{S}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times \boldsymbol{M''}$$

$$\boldsymbol{S'} = \sqrt{\boldsymbol{S}_x^2 + \boldsymbol{S}_y^2}$$

iii.

1. Sx and Sy denote the horizontal and vertical edge convolutions. S' denotes the gradient magnitude which in my implementation I calculated as $|S_x + S_y|$ since that yields effectively the same results.

$$\mathbf{U} = \boldsymbol{Z}\left(\boldsymbol{G}\boldsymbol{S'}\right)$$

iv.

1. The above formula denotes a thinning algorithm described within the UnCanny Filter. I decided not to implement this portion of the method as I had initialized the algorithm using the edges detected from the frames as opposed to a raw framewise difference.

Section III: Results and Future Work

Through the development of an image processing toolbox, I was able to expand my knowledge of the mathematical and algorithmic framework within which images can be understood and manipulated. The single-pixel operations, histogram equalization, and order-statistic filtering allowed me to craft an idea of the impact of objective function parameters on largely subjective image enhancement effects as well as denoising operations. Edge detection and my experiment implementing the UnCanny Filter allowed me to explore my areas of interest in the cross-section between art and computer processing, i.e Generative Art (Honeycutt and Bridge, 2021). My initial interest in this project was largely driven by an existing program I had explored called EbSynth.

EbSynth employs a technique known as example-based synthesis in order to map painted or artistically rendered frames of a video onto the real video frames. This can be illustrated using the figure 1 and 2 below taken from EbSynth's sample project.

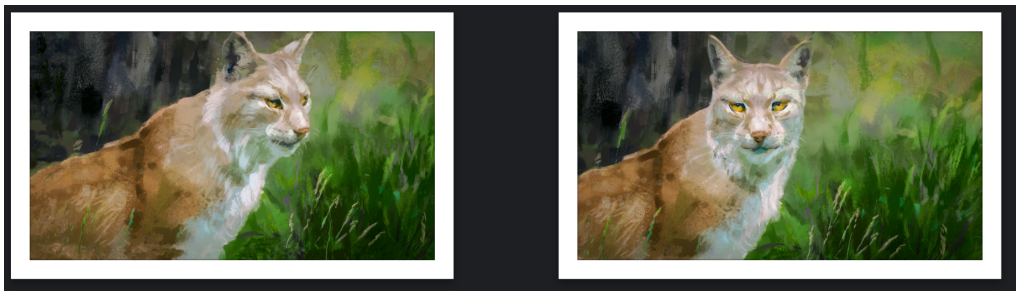


Figure 1. The artistic rendition of frame 0 and frame 99 of the video.

Figure 2. A subset of the frames of the captured video which Ebsynth will use to produce the artistic renditions of the missing frames.The information I was initially able to find about the functionality behind the program was limited to non-technical

articles which described it as dividing up the existing images into portions and mapping them onto the artistically rendered textures (Sochorová, 2022). This immediately brought to mind image segmentation, but after further research I concluded that it would be more productive to first grasp and develop an efficient way of tracking motion between video frames. I decided to implement Sobel Edge Detection and then went on to implement the UnCanny Filter. I experimented with different ways of applying the UnCanny Filter and compared results in order to settle on the algorithm that seemed most productive to what I plan on developing next. The following figure illustrates my findings:

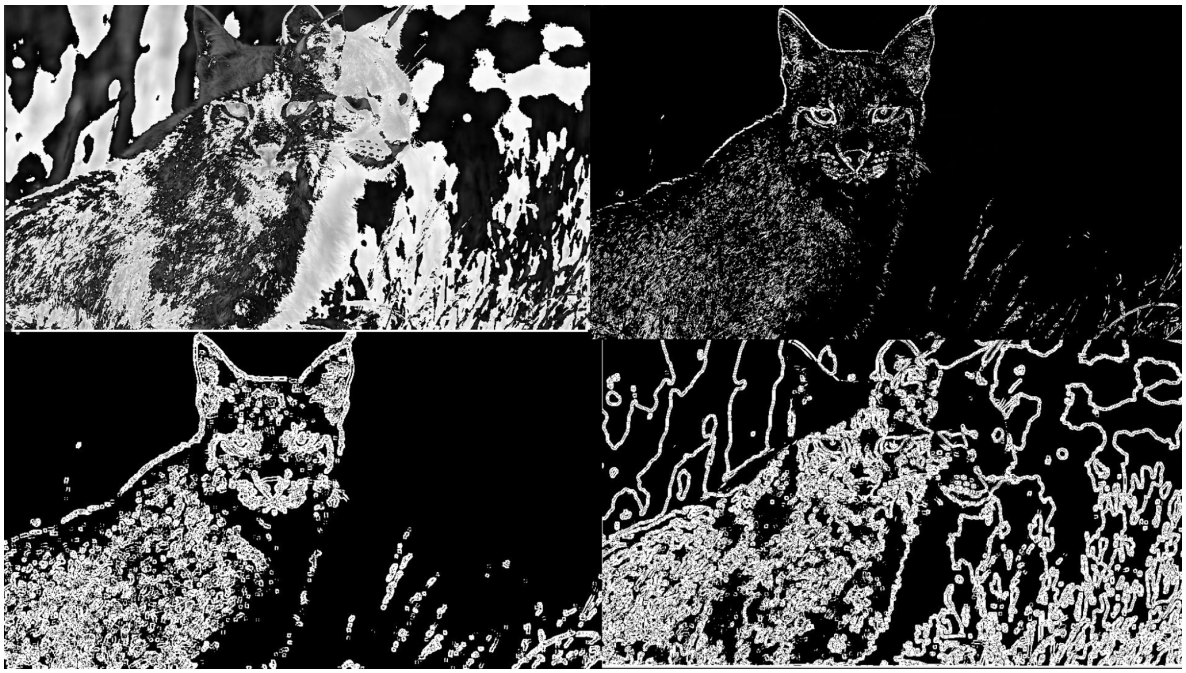


Figure 3. Framewise difference, i.e, video frame 2 - video frame 1 (top left), Difference between binary edge maps (top right). Binary edge maps and Uncanny Filter (bottom left), UnCanny Filter only (bottom right).

The framewise difference and the uncanny filter alone (top left and bottom right) seem to have produced the least optimal results for reproducing the subsequent frame. The binary edge map difference alone and combined with the UnCanny Filter (top right and bottom left) seem to have produced the more optimal results.

At this stage it is unclear if there is an advantage to applying the UnCanny Filter over binary edge mapping alone; however, it should become more clear what provides the most accurate results as I begin developing the functionality of applying the artistically rendered textures to the produced frames. Through continued research and experimenting with different methods and techniques, I have been able to outline my next steps. I would like to implement image segmentation, most likely using edge detection as well as explore methods of image style transfer in order to produce the textured images.

References

Gonzalez, Rafael C, and Richard E. Woods “Digital Image Processing.” Pearson Education. 2018 pp.123.

Honeycutt, Wesley T, and Eli S Bridge. “UnCanny: Exploiting Reversed Edge Detection as a Basis for Object Tracking in Video.” Journal of imaging vol. 7,5 77. 23 Apr. 2021, doi:10.3390/jimaging7050077

Sochorová, Šárka. "Bringing Pictures to Life in EbSynth." *80.Lv*, 14 Feb. 2022, 80.lv/articles/bringing-pictures-to-life-in-ebsynth/.