Tutorial 1

Gidon Rosalki

2025-10-22

1 Agenda

1. Introduction
2. Images
3. Intensity transformations
4. Histogram equalisation
5. Histogram matching
6. Exercise 1

2 Introduction

There will be 5 individual programming exercises, all of which must be submitted and passed.
There will be 3 exams, of 1 hour in length, at 1800 on 2025-11-16, 2026-12-14, and 2026-01-21 . They will be open book, with no resits.

3 Images

What is an image? There are 2 main types of pictures, RGB, and greyscale. A greyscale image is essentially a matrix, where each pixel is an integer from 0 to 255 (usually). In an RGB image, each pixel is made up of 3 unsigned bytes (same as greyscale), each one representing one of the colours red, green, and blue. The more pixels we have in the image, usually the greater detail is visible in the image, but the more space it will require.

For greyscale we may represent each pixel as a float between 0 and 1, instead of a uint8. Some applications (GPUs) find it easier to work with these kinds of floats. Binary images are boolean matrices of 0s and 1s (see exercise 3). Finally RGB images have 3 greyscale channels, each one representing one of the colours red, green, and blue.

In addition to the RGB colour space, there are many more colour spaces, such as the YIQ colour space. Here, Y is the greyscale channel, and together I and Q encode the colours by the following calculation:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

This is generally used in TVs, and came about to provide backward compatibility to greyscale TVs (that only used the Y channel), with colour TVs.
There is also HSB - Hue Saturation Brightness colour space, often used in photo editing, since it is easier for humans to use to get a precise colour.

4 Intensity transformations

Our first case is the single pixel intensity transformation. In this case, we can represent this function with a lookup table:

$$s = T(r)$$

Where r is the old pixel, s is the new, and it is transformed through the function / table T. For example, consider the function

$$T(r) = 255 \cdot \left(\frac{r}{255}\right)^{0.3}$$

This function has its problems, since we firstly divide r by 255, then do something to it, and then remultiply it. Instead, if we work with $s, r \in [0,1]$ then we can simply perform the function

$$s = r^{0.3}$$

Much simpler.

Let us also consider the function

$$r \in [0,1] \;\; T(r) = 1 - r$$

This inverts the image. This is often very useful in the medical imaging world.

Another useful functionn is the log transform, which increases the visibility of a very dark image.

$$\begin{aligned} T(r) &= c \cdot \log(1+r) \\ c &= \frac{255}{\log(1+\text{maxInputVal})} \\ s, r &\in \{0, \ldots, 255\} \end{aligned}$$

We also have the family of γ correction functions where

$$T(r) = c \cdot r^{\gamma}$$

These are useful for correcting the colours of an image. For example, a projector's image is on a grey background. Since the image is expecting a white background, so a correction needs to be applied such that the image is brightened. A gamma larger than one will darken the image, where a gamma smaller than one will brighten the image.

5 Histogram equalisation

What is a histogram? A histogram is a graph that measures how many pixels there are for each level of greyscale (and sometimes divides by how many there are in total. Both will be used). We also have cumulative histograms, which show how many pixels were in the image until this greyscale value (monotonically increasing).

Our objective with histogram equalisation is to improve the image contrast, by making equal use of all grey levels. We would like a histogram that is uniformly distributed. This will not happen, but we can dream. This is effectively the same as aiming for a cumulative histogram, described by the function $f(x) = x$.

In order to compute the image histogram, we have wonderful numpy functions, such as `np.histogram`. To find the cumulative histogram we can use `np.cumsum`. WE then normalise the cumulative histogram (by dividing by the total number of pixels), and then multiply the normalised histogram by the maximal grey level value $(Z-1)$. Next we need to veriify that the minimum value is 0, and maximum is $z = 1$, otherwise stretch the result linearly in the range $[0, Z-1]$. Round the values to get integers, and then map the intensity values of the image using the result of the previous step. The algorithm is, take the cumulative histogram $C(k)$, and let m be the first greyscale level, such that $C(m) \neq 0$. Then

$$T(k) = \text{round}\left(255 \cdot \frac{C(k) - C(m)}{C(255) - C(m)}\right)$$

In most discrete images it is impossible to achieve a completely uniform histogram – only an approximation. The histogram equalization transformation is monotonic (as the process involves the cumulative sum), and therefore the relative brightness of a pixel is preserved. The number of different pixel values (full bins in the histogram) can only decrease. We might merge bins, but we cannot split them.

This will fail when the desired grey level distribution is not uniform (consider a text document). It will also fail when content from different sources is equalised together (such as from two non related images).

6 Histogram matching

Histogram matching is a process to adjust the histogram of a source image, to resemble the histogram of a target image. It is used to transfer appearance characteristics (such as brightness, and contrast) from one image to another. The steps are as follows:

1. Compute the histograms of the source, and the target
2. Calculate the Cumulative Distribution Function (CDF) for both histograms
3. Match intensities by mapping pixel values from the source image, to the corresponding values in the target histogram, using the CDFs
4. Transform the image based off the mapping