Template Matching

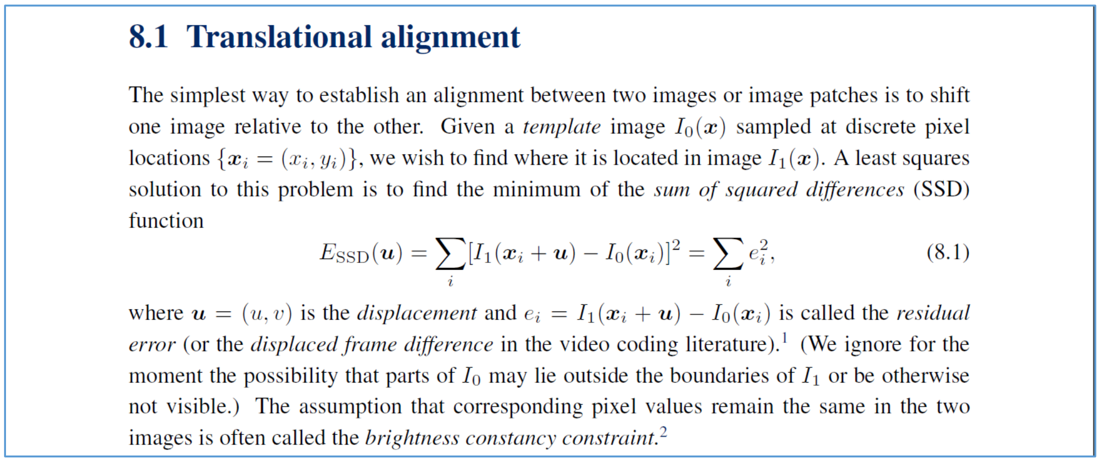
The purpose of template matching is to search for patterns in an image that are similar to the template. In its simplest form we could search for an exact match but this would be of limited use since the RGB values might be significantly different for the same object appearing in different images under different lighting conditions or camera exposure. Foremost, we would like the matching procedure to be independent of the brightness. We will explore this case further in this course. In the more general case, we would like the search to be tolerant to scale and rotation (and other deformations) as necessary.

In template matching, the user determines the content of the template and provides a small image as the query. For example, if we want to find car wheels in images we would need to find an image with a good wheel prototype and cut it out of the image and use it as the template.

In the following discussion, to keep it simple, let us assume that the image and the template are grayscale images.

Matching with intensity differences

A simple way to measure similarity between a template and a piece of the image (chunk) is to find the minima of the sum of squared intensity differences:



8.1 Translational alignment

The simplest way to establish an alignment between two images or image patches is to shift one image relative to the other. Given a *template* image $I_0(\boldsymbol{x})$ sampled at discrete pixel locations $\{\boldsymbol{x}_i = (x_i, y_i)\}$, we wish to find where it is located in image $I_1(\boldsymbol{x})$. A least squares solution to this problem is to find the minimum of the *sum of squared differences* (SSD) function

$$E_{\text{SSD}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 = \sum_i e_i^2, \tag{8.1}$$

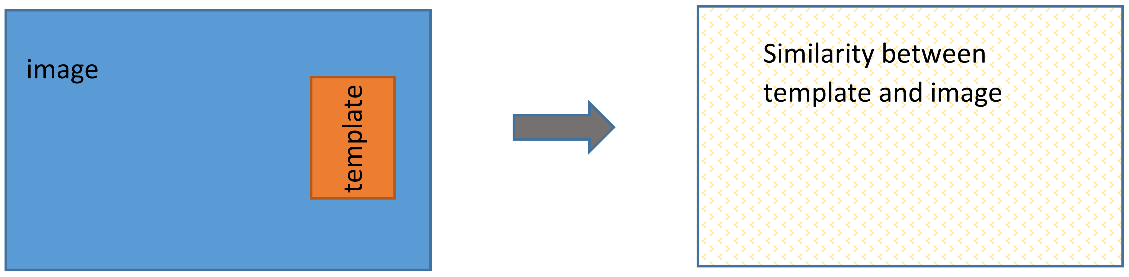
where $\boldsymbol{u} = (u, v)$ is the *displacement* and $e_i = I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)$ is called the *residual error* (or the *displaced frame difference* in the video coding literature).[1] (We ignore for the moment the possibility that parts of I_0 may lie outside the boundaries of I_1 or be otherwise not visible.) The assumption that corresponding pixel values remain the same in the two images is often called the *brightness constancy constraint.*[2]

Szeliski p. 384

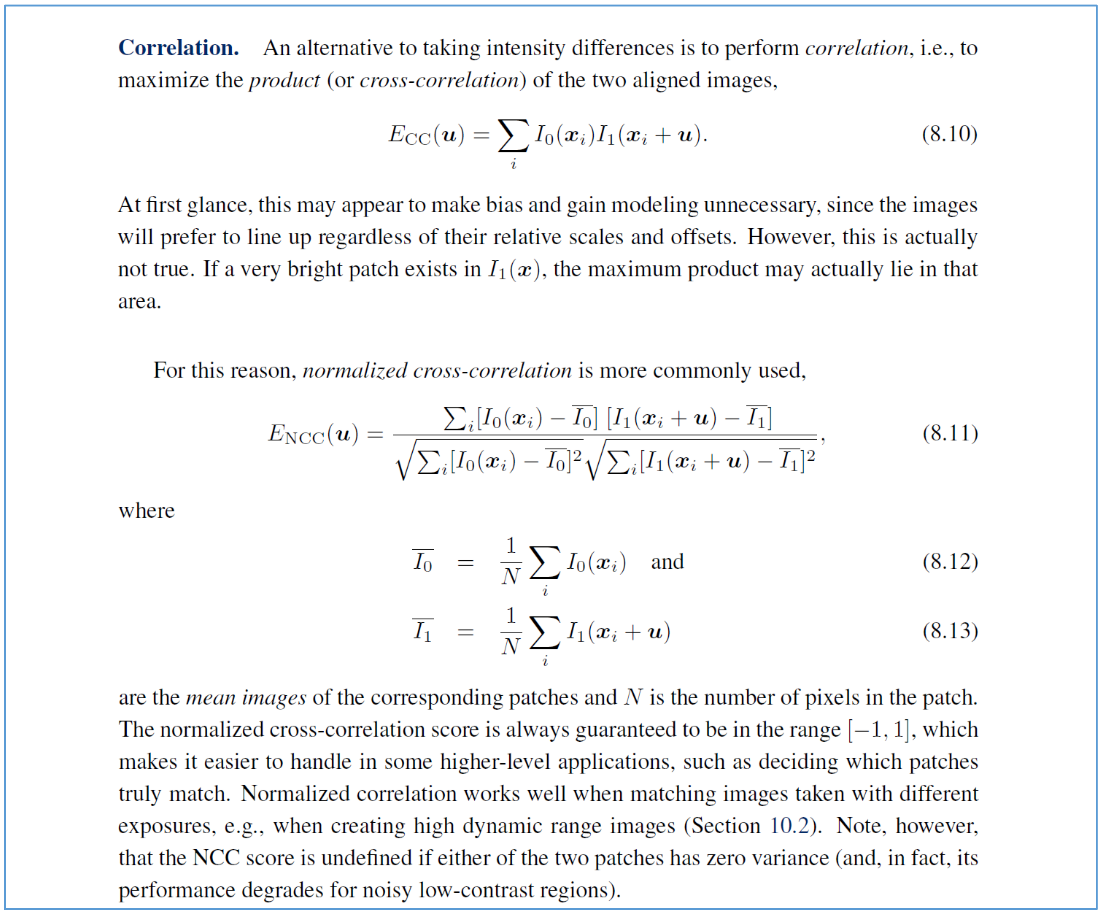
This, however, will work well with exact matches but not very well with approximate matches.

Filters as Templates

We can think of a template as a rectangular filter kernel and apply our filtering algorithm from before. The result of filtering will produce a matrix containing similarity values between the template and the image chunk. We can then report the most similar locations as the positions of the matches.



In our neighborhood processing implementation, we have used (unnormalized) correlation to calculate our filter's outputs.



Correlation. An alternative to taking intensity differences is to perform *correlation*, i.e., to maximize the *product* (or *cross-correlation*) of the two aligned images,

$$E_{\mathrm{CC}}(\boldsymbol{u}) = \sum_i I_0(\boldsymbol{x}_i) I_1(\boldsymbol{x}_i + \boldsymbol{u}). \tag{8.10}$$

At first glance, this may appear to make bias and gain modeling unnecessary, since the images will prefer to line up regardless of their relative scales and offsets. However, this is actually not true. If a very bright patch exists in $I_1(\boldsymbol{x})$, the maximum product may actually lie in that area.

For this reason, *normalized cross-correlation* is more commonly used,

$$E_{\mathrm{NCC}}(\boldsymbol{u}) = \frac{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]\,[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2}\sqrt{\sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]^2}}, \tag{8.11}$$

where

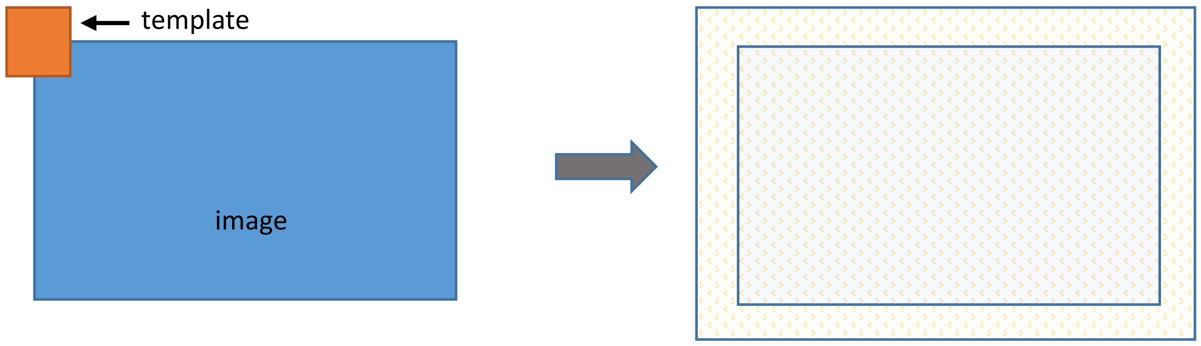
$$\overline{I_0} = \frac{1}{N}\sum_i I_0(\boldsymbol{x}_i) \quad \text{and} \tag{8.12}$$

$$\overline{I_1} = \frac{1}{N}\sum_i I_1(\boldsymbol{x}_i + \boldsymbol{u}) \tag{8.13}$$

are the *mean images* of the corresponding patches and N is the number of pixels in the patch. The normalized cross-correlation score is always guaranteed to be in the range $[-1, 1]$, which makes it easier to handle in some higher-level applications, such as deciding which patches truly match. Normalized correlation works well when matching images taken with different exposures, e.g., when creating high dynamic range images (Section 10.2). Note, however, that the NCC score is undefined if either of the two patches has zero variance (and, in fact, its performance degrades for noisy low-contrast regions).

Szeliski p. 387

MATLAB has a function called **normxcorr2** that implements normalized correlation.



Matlab's normalized correlation function extends the borders and returns a normalized correlation matrix that has size equal to the input image plus the template (minus 1)

e.g.
template: 73 55
image: 600 900
norm corr: 672 954

Search algorithm for a template with fixed size and one orientation:

```
Load image $\mathbf{I_1}$ and template $\mathbf{I_0}$
Set a threshold (<= 1)
Calculate normalized correlation NC between image $\mathbf{I_1}$ and the template $\mathbf{I_0}$ for all possible positions
While NC contains elements >= threshold
        Find the largest peak in NC, record the coordinates if greater than the threshold
        Zero out a neighborhood of the peak in NC
Output the list of coordinates
```

Search algorithm to detect matches for different orientations:

```
Load image $\mathbf{I_1}$ and template $\mathbf{I_0}$
Set a threshold (<= 1)
For all possible orientations
        Rotate $\mathbf{I_0}$ and save in $\mathbf{I_R}$
        Calculate normalized correlation c between image $\mathbf{I_1}$ and $\mathbf{I_R}$
        While NC contains elements >= threshold
            Find the largest peak in the normalized correlation function, record the coordinates
                and rotation if greater than the threshold
            Zero out a neighborhood of the peak in NC
Output the list of coordinates and corresponding orientations
```

To search for different sizes and orientations at the same time, another nested loop can be added to try all possible scales.

Templates for Non-rectangular Objects

The above discussion assumes that we can find rectangular regions to characterize an object and cut out these regions from an image to use as a template. This works only if we can identify a small rectangle that is unique to the object or if the background of the image (in which we are searching the template) is the same as that of the template. For this method to work with arbitrary backgrounds we will need to identify the pixels of the object and apply the approach discussed above only to those pixels.