Registration Optimization

L33

Goal: To see what factors influence registration optimization, and how we can speed registration up.

Interpolation Effect

When an image is interpolated, it often causes a bit of information loss, often in the form of blurring. The histogram of a blurred image is more **spread out,** causing the entropy to **increase.** Hence, we have:

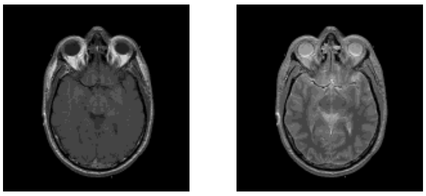
- "information loss" in the sense that we cannot exactly reconstruct our original image because of aliasing, and
- "information gain" in the snese that the entropy has increased.

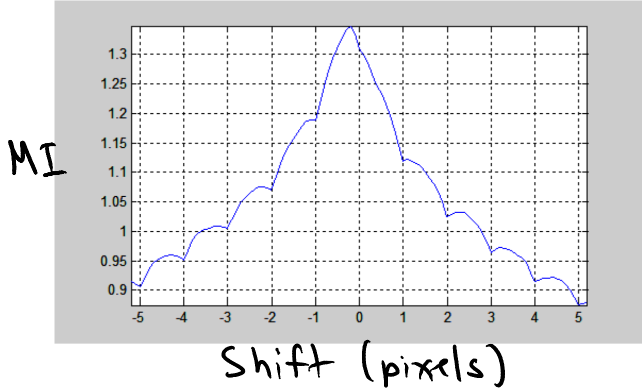
This dichotomy is meaningless; I just point it out to avoid confusion cuase by the word "information".

Thus, when one image is resampled, its marginal entropy **increases**, so the MI also **increases** slightly. Integer pixel shifts do NOT result in interpolation, but fractional pixel shifts DO.

The interpolation effect causes the cost function to be wavey. This sampling artifact can be a problem for registration, since it can create lots of local optima.

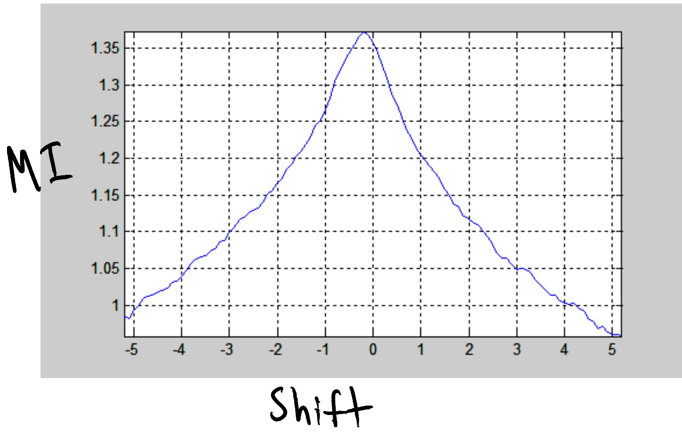
(sweet Matlab demo)





In this example, the MI cost function prefers fractional shifts over integer shifts.

One way to combat this issue is to force **interpolation** of both images. For example, we can start out by rotating both images by 45 .



This "interpolation effect" can also influence other cost functions. So be aware of it.

Nelder-Mead Optimization

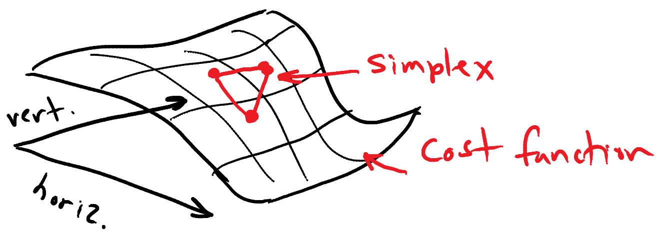
Many optimization methods use derivatives.

- Gradient descent
- Our SSD method used a linear approximation

Not all optimization methods use derivatives. The Nelder-Mead method uses only function values of the cost function. It sets up a **"simplex"** of points in the parameter space where the cost function is evaluated. Then, these points are moved - one or two at a time - to crawl toward a better (higher or lower) solution. The simplex consists of $p+1$ points, where p is the number of parameters you are optimizing over.

For example, suppose we are looking for the optimal horizontal and vertical shift.

The simplex has 3 points, with initial positions specified manually.



These 3 points are labelled:

w = worst

b = best

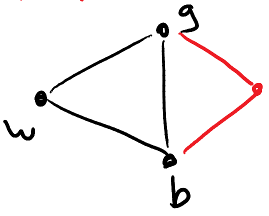
g = good

Then we change the simplex using these rules:

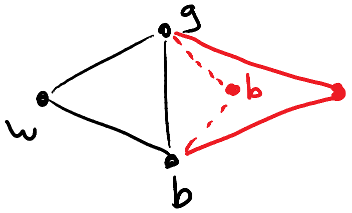
1. Reflect *w* through the centroid (centre) of the simplex
   a. If this produces a new "best", then create a new simplex by **extending** the simplex further. Go to 1.
   b. If this produces a "good" point, relabel the points. Go to 1.
   c. If this produces a new "worst", then create a new simplex by **un-reflecting** and **compressing** toward the best point. Go to 1.
2. Continue until the simplex is small enough, or the difference between the best and the worst is small enough.

In pictures:

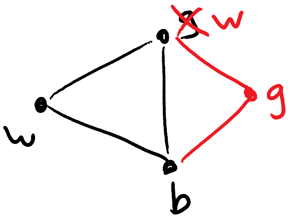
1) Reflect



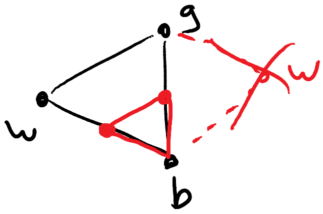
a) If best, extend further, go to 1



b) If good, go to 1



c) If worst, then compress, go to 1



Matlab's built-in "fminsearch" method is an implementation of Nelder-Mead.

Multiresolution Approaches

This refers to solving a **lower-resolution** (and often **smaller**) version of the problem first, and then using its result as a starting point for a **higher-resolution** (and usually more **expensive**) version.

This is done for two reasons:

Coarse Adjustments

On a coarse scale, adjustments are based on larger-scale structures, so there is less tendency to get stuck on **details**. Similarly, a one-pixel step in $\frac{1}{4}$ resolution is the same as a **4-pixel** step in full resolution. Hence correction of gross displacements is rapid.

Computational Complexity

Reduced-scale images have fewer pixels, so pose less of a computational burden.

These two factors complement each other. They allow for quick convergence on a coarse scale.

(yet another amazing Matlab demo)