Bioengineering Imaging Track Qualifying Exam Study Book

Markus Foote and Blake Zimmerman

August 11, 2017

Contents

1. Affine Transform

An *affine transform* is the combination of a linear transformation with a translation. In a linear algebra contest, a point in space is a column vector of individual values along each dimension, i.e. $\begin{bmatrix} x \\ y \end{bmatrix}$ or $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ in 2 or 3 dimensions, respectively. This location is noted simply as $\vec{x}$.

The affine transform can thus be written as a matrix-vector multiplication, followed by vector addition:

$$\vec{y} = \mathbf{A}\vec{x} + \vec{b} \tag{1.1}$$

The linear transformation matrix $\mathbf{A}$ has some important special cases:

Scaling This transformation scales each direction of a vector space by the corresponding c value:

$$\mathbf{A} = \begin{bmatrix} c_x & 0 \\ 0 & c_y \end{bmatrix} \text{ or } \begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & c_z \end{bmatrix}$$

Rotation This transformation rotates by an angle θ about the origin in 2 dimensions:

$$\mathbf{A} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Translation This only translates all the points by a constant amount:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } b = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Skew

Others?

These transformations are all well and good, but with them so far we can only take points and apply some fun tricks to move them around in cute ways. What if there were pairs of points with *known* (or *assumed*) correspondence, and we want to find an affine transform that makes the point pairs match?

Using notation:

$$\vec{x}_i \quad \text{source point(s)} \tag{1.2}$$
$$\vec{y}_i \quad \text{target point(s)} \tag{1.3}$$
$$n \quad \text{total number of point pairs} \tag{1.4}$$
$$\mathbf{A} \quad \text{linear transform matrix} \tag{1.5}$$
$$T \quad \text{translation vector} \tag{1.6}$$

In a perfect world, all the points match exactly after being transformed, such that

$$\vec{y}_i = \mathbf{A}\vec{x}_i + T \tag{1.7}$$

is satisfied. This linear system of equations could be easily solved with some linear algebra.

However, this world is not perfect. Due to noise, human error, systematic error, and spilling coffee on samples before *carefully* wiping it up, there will be noise and the points will not exactly match. There will be some error. We must account for this error.

We want to estimate the best transformation given some number of point pairs that will not all match exactly. Thus, we should minimize the error in the matching using a euclidean distance for the error, and squared error so that this function is convex (which is important when very close to zero error):

$$\min_{\mathbf{A},T} \sum_i \|(\mathbf{A}\vec{x}_i + T) - \vec{y}_i\|^2 \tag{1.8}$$

How do we solve this?

First, if we are given $\mathbf{A}$, then

$$\delta T = 2\sum_i^n (\mathbf{A}\vec{x}_i + T - \vec{y}_i) = 0 \tag{1.9}$$

The variation of T is set to zero because the first derivative of a function is zero at a minimum. Solving for T:

$$T = \frac{1}{n}\sum_{i=1}^{N} (\vec{y}_i - \mathbf{A}\vec{x}_i) \tag{1.10}$$

This translation is simply the difference of the centroids of the source and target points. Let us define

$$\tilde{y}_i = \vec{y}_i - \vec{c}_y \tag{1.11}$$

$$\tilde{x}_i = \vec{x}_i - \vec{c}_x \tag{1.12}$$

where c_x and c_y are the centroids of each point set. This modification of the problem implicitly removes T since both point sets will have the same centroid (the origin).

Assuming the points are centered the problem

$$\min_{\mathbf{A}} \sum_i \left\| \mathbf{A}\left(\vec{x}_i - \frac{1}{n}\sum_j \vec{x}_j\right) - \left(\vec{y}_i - \frac{1}{n}\sum_j \vec{y}_j\right) \right\|^2 \tag{1.13}$$

simplifies to

$$\min_{\mathbf{A}} \sum_i \|\mathbf{A}\tilde{x}_i - \tilde{y}_i\|^2 \tag{1.14}$$

Similar to the procedure performed to find and remove T, we now find the variation of $\mathbf{A}$ and set it equal to zero:

$$\frac{\partial}{\partial \mathbf{A}} = 2\sum_i (\mathbf{A}\tilde{x}_i - \tilde{y}_i)\ \tilde{x}_i^T = 0 \tag{1.15}$$

Then solving for $\mathbf{A}$:

$$\mathbf{A} = \left(\sum_i \tilde{x}_i \tilde{x}_i^T\right)^{-1} \left(\sum_i \tilde{y}_i \tilde{x}_i^T\right) \tag{1.16}$$

A special note here about the matrix $\sum_i \tilde{x}_i \tilde{x}_i^T$: A minimum of 3 non-co-linear points in 2D (or 4 non-coplanar points in 3D) are required for this matrix to be invertible.

2. Magnetic Resonance Imaging

3. Image Processing: Hough Transform

In image processing the **Hough Transform** is useful for *global* filtering, especially to find sets of pixels that lie along curves of a specified shape, i.e. finding lines, circles, ellipses, spheres, hypercubes, etc. It provides an elegant solution using a *discretized parameter space*, while the naïve, brute-force approach quickly becomes daunting. In the case of finding lines in an image of n points, the naïve approach involves iteration over all $\sim n^2$ possible lines and performing $\sim n^3$ comparisons for every point to each line. The $O(n^3)$ complexity exponentially worsens for shapes with higher dimensional parameter spaces. This approach is computationally prohibitive for non-trivial applications.

Instead, the Hough Transform accumulates non-background points in a discretized parameter space. The dimensionality of the parameter space is equal to the number of the parameters that describe the desired shape. In the case of a line, there are two parameters. However, the parameters must be defined with care. The naïve parameter choices for a line might be *slope* and *intercept*, but if the objective might include finding vertical (or nearly vertical) lines, a method to accurately discretize/store infinity (or very large values) would be required. A better approach is to parameterize lines by their *normal* representation:

$$x\cos(\theta) + y\sin(\theta) = \rho\ . \tag{3.1}$$

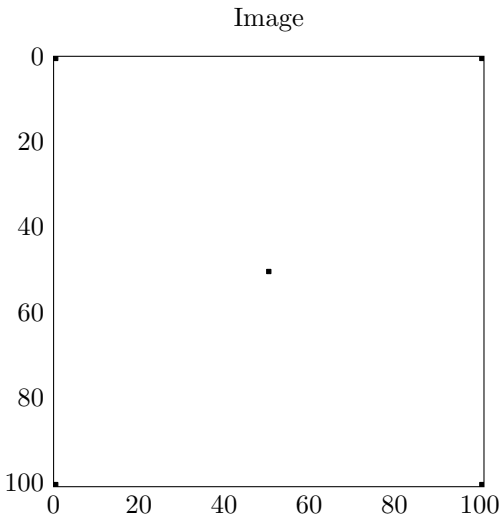
In this parameterization, all parameters are nicely bounded: $-\pi/2 \leq \theta \leq \pi/2$ and $-D \leq \rho \leq D$ for an image with D length between the two most distant corners.

The Hough Transform is a general algorithm, though in some cases it is analogous to formal mathematical transforms (eg. the Hough transform of lines with 'normal' parameterization is analogous to the Radon Transform). A typical pre-processing step for the image in question is edge detection to form a binary image, though this is not strictly necessary. The algorithm proceeds as follows:

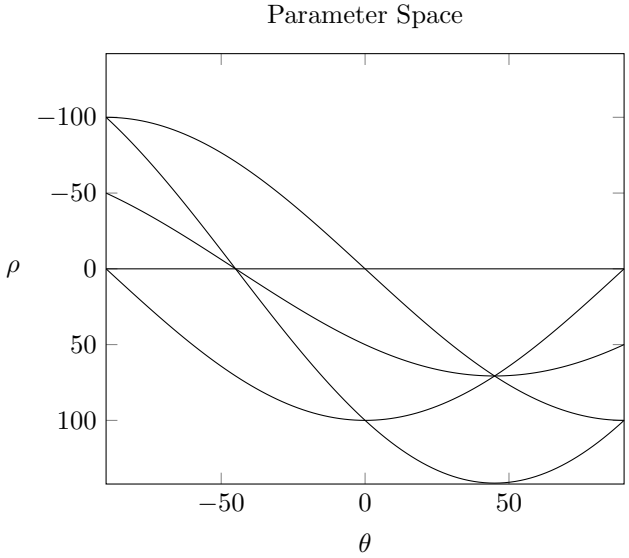
1. Allocate an accumulator image, selecting the appropriate number and width of bins for each parameter.
2. Select a foreground pixel.
3. Select center-of-bin values for (the same) $n-1$ of the n parameters.
4. Solve for the final parameter value to satisfy the reference equation at the selected pixel.
5. Increment the accumulator image at the chosen- and solved- parameter location.
6. Repeat steps 3 - 5 for all possible chosen parameter values.
7. Repeat steps 2 - 6 for all foreground pixels in the image.
8. Find parameters for prominent features by the location of accumulator maximum, or thresholding.

In the case of non-binary images, the accumulator simply need not be discrete, such that foreground (or all) pixels just have a partial-voting effect. This algorithm simply extends to higher dimensional images, and higher dimensional parameter spaces. In general, the complexity depends on the number of foreground pixels and the number of parameters; the complexity *increases* by $O(A^{m-2})$ with each additional parameter, where A is the size of the image space, and m is the number of parameters.

As a concrete example, consider the binary image (with foreground as black)



By performing a Hough Transform with (3.1) defining the parameters, and with the range of $\theta = \pm 90°$ and $\rho = \pm\sqrt{2}C$, the resulting accumulator image is



(Realistically these sinusoids are just the patterns of the discrete bins, not actual sinusoidal curves, but I'm lazy and made 5 curves instead of 101^2 boxes.) The locations where these curves intersect have accumulator values of 2 or 3 (however many curves intersect). From the intersection of the three curves at $(45, 70.7)$ and $(-45, 0)$, we conclude that there are three foreground points that lie on each lines diagonally across the image. Also notice how this special case of the Hough Transform is equivalent to the Radon Transform.

As a brief peek at generalization, the Hough Transform for a circle might be based on the parameterization

$$(x-a)^2 + (y-b)^2 = r^2 \tag{3.2}$$

where the center lies at (a, b) with radius r. With three parameters, this algorithm is significantly more complex, though it can be decreased by knowing the radius of circle which is desired *a priori*. Additionally, finding a circle presents interesting considerations, such as the *center* of the circle not being within the original image space while the pixels that contribute to that circle do.

4. Computed Tomography: Physics and Reconstruction

4.1 X-ray Summary

4.2 Rudimentary Projection Experiment Setup

4.3 Reconstruction from Projection Data