

We have spent several lectures on how to process images and detect features such as edges and boxes, and estimate image transformations such as translations. Throughout the rest of this course, we will use these image measurements to estimate the parameters of some of the models that we saw in the first part of the course, namely models of external and internal parameters, and scene geometry. We will use several estimation techniques. Today we look at three popular ones: least squares, Hough transforms, and RANSAC.

*The ordering of material in these notes is slightly different from the slides. In these notes, the vanishing point example is presented only at the end.*

## Least squares line fitting

If you have taken a statistics course, then you have seen the following version of the line fitting problem. You have two data variables  $x$  and  $y$  and you want to fit a linear relationship between samples  $(x_i, y_i)$  such that

$$y_i = mx_i + b. \quad (1)$$

Typically this model does not fit the data exactly because of noise or other variability and, in particular, the variability is in  $y$  only, e.g. the  $x$  variable might be chosen by the experimenter (and is not random) and so only the  $y$  is random. In this case, one assumes a model

$$y_i = mx_i + b + n_i$$

where the  $n_i$  are noise. To fit the line, you find the  $m$  and  $b$  that minimizes

$$\sum_i (y_i - mx_i + b)^2 = \sum_i n_i^2.$$

To do so, you take the partial derivatives with respect to  $m$  and  $b$  and set them to 0. This gives you two linear equations with two unknowns  $m$  and  $b$  and coefficients that depends on the data  $x_i$  and  $y_i$ . You can easily solve these equations to get  $m$  and  $b$ .

Let's look at a slightly different version of the problem which is more common in vision. Again we have a set of points  $(x_i, y_i)$  in a 2D plane and we want to fit a line to these points. However, rather than taking our noise or "error" to be in the  $y$  direction only, we take the error to be the distance from  $(x_i, y_i)$  to a line

$$x \cos \theta + y \sin \theta = r \quad (2)$$

where  $\theta$  is the direction of the *normal* to the line and  $r$  is the perpendicular distance from the origin to the line in direction  $\theta$ . Note Eq. (2) is a more general equation for a line than Eq. 1 since Eq. (2) allows for lines of the form  $x = \text{constant}$ .

The perpendicular distance from any point  $(x_i, y_i)$  in the plane to such a line is:

$$|x_i \cos \theta + y_i \sin \theta - r|.$$

We want to find the  $\theta$  and  $r$  that minimize the sum of squared distances. (This is called *total least squares*.)

How do we solve for  $\theta$  and  $r$ ? We cannot just use the same method as above, i.e. try to minimize

$$\sum_i (x_i \cos \theta + y_i \sin \theta - r)^2$$

directly, because you have the non-linearity of the cosine and sine. (As you vary  $\theta$ , for any fixed  $r$ , the sum of squares expression is not a simple quadratic as before.) Instead, we find the values of  $a$ ,  $b$  and  $r$  that minimize

$$\sum_i (x_i a + y_i b - r)^2 \quad (3)$$

subject to the (non-linear) constraint that  $a^2 + b^2 = 1$ , that is,  $(a, b)$  is a unit vector. This means we have a *constrained minimization* problem.

Taking the derivative of (3) with respect to  $r$  and set it to 0 gives:

$$a \sum_i x_i + b \sum_i y_i = r \sum_i 1.$$

Letting  $\bar{x} = \frac{\sum_i x_i}{\sum_i 1}$  and  $\bar{y} = \frac{\sum_i y_i}{\sum_i 1}$  be the sample means, we get

$$a\bar{x} + b\bar{y} = r$$

which says that the sample mean  $(\bar{x}, \bar{y})$  falls on the solution line. Substituting  $r$  into (3) gives

$$\sum_i ((x_i - \bar{x})a + (y_i - \bar{y})b)^2.$$

Recall we are trying to minimize this expression, subject to  $a^2 + b^2 = 1$ . But we can write this expression differently, namely define an  $n \times 2$  matrix  $\mathbf{A}$  whose rows are  $(x_i - \bar{x}, y_i - \bar{y})$ . The expression becomes  $(a, b)\mathbf{A}^T \mathbf{A}(a, b)^T$ . For  $a^2 + b^2 = 1$ , the expression is minimized by the eigenvector of  $\mathbf{A}^T \mathbf{A}$  having the smaller eigenvalue. (Note both eigenvalues are non-negative since  $\mathbf{v}^T \mathbf{A}^T \mathbf{A} \mathbf{v} \geq 0$  for any  $\mathbf{v}$ .) Thus, our solution amounts to considering all the lines that pass through  $(\bar{x}, \bar{y})$ , and seeing which perpendicular direction  $\theta$  minimizes the sum of square distances to the points  $(x_i, y_i)$

## Inliers and outliers

One problem with least squares techniques is that it (implicitly) assumes that the noise distribution is the same for the all points. In many situations, though, the situation is more complicated, namely there are some points that obey the model (called *inliers*) and other points that do not (called *outliers*). In the line fitting problem, if we are penalizing all points by the squared distances to a candidate line, the outliers can have a huge penalty. This can drive the estimated solution away from the correct solution (that is, the correct solution *for the inliers*).

Let's now look at a few other methods that are more robust to outliers and that are popular in computer vision, namely the Hough transform and RANSAC.

## Hough transform

For each sample point  $(x_i, y_i)$  we are penalizing each line  $(\theta, r)$  by adding a penalty that is the squared distance from the point to that line. There are other possibilities, though. We could penalize points by their squared distance from the line (or their absolute distance) *up to some distance*, say  $d_{max}$ . and then beyond this distance, have a constant error. There are many methods based on this idea and they fall under the name *robust statistics*. The simplest such approach is to give zero error up to some margin distance  $\tau$  and then constant error beyond that. We would then try to find the  $(\theta, r)$  that minimizes this error. For example, one could use the simple algorithm:

```

for each line (theta,r) // need to choose a sampling (quantization)
  Count number of points (xi,yi) that fall outside the distance tau from line
return the (theta,r) pair with the smallest count

```

The Hough transform does essentially what I just described except that it counts (votes for) points *within* the  $\tau$  margin, rather than counting the bad points that lie outside the margin.

```

for each (x_i,y_i){
  for each theta{ // need to choose a sampling
    r := round(x_i cos(theta) + y_i (sin theta))
    Vote for (r,theta)
  }
}
return the (theta,r) pair with the most votes

```

Minor technical point: for a given  $(x_i, y_i)$ , the line  $(r, \theta)$  is identical to the line  $(-r, \theta + \pi)$ . Thus the votes would always have a certain two-fold symmetry. One would choose the solution with  $r > 0$ .

The transformation from a set of points  $(x_i, y_i)$  to a histogram of votes in the model parameter space (in this case, a 2D grid  $(r, \theta)$ ) is known as the *Hough Transform*.

The advantage of Hough is that the outliers have very little effect on the estimate. As long as the outliers don't conspire to vote on some other particular model, the votes of the outliers will be spread out over the  $(r, \theta)$  space. Of course, if there really two good line models present to explain the data, then you will get two peaks and you would need to choose between them, or just take both and conclude there are two models.

## RANSAC (Random Sample Consensus)

Let's now consider second approach which is often used in computer vision when there are lots of outliers and when the number of parameters of the model is more than two. We will see examples later when the number of parameters is 7 or 8, for example, and you want to estimate these parameters precisely. In this case, a Hough transform approach just doesn't work. For now, you can just think of the same problem as above, namely fitting a line model to a set of points.

We have seen that least squares tries to use all of the data to fit a model. This next approach (RANSAC) is the extreme opposite. It fits a model using the minimal number of points possible. For a line, the minimal number of points is 2.

The RANSAC algorithm for line fitting is roughly as follows. (There are several versions.) Sample a large number of point pairs. For each point pair, fit a line model, namely find the unique line passing through the two points. Check all the other points and see how many of them lie near the line, where "near" means within some distance threshold  $\tau_1$ ). These points are called the *consensus set* for that line model. Repeat this some number of times. Then, choose the model that has the largest consensus set. Use least squares to fit a model to this consensus set and terminate.

At first glance, this algorithm seems a bit nutty, since your intuition is that even if you happen to sample only inliers, each of the points has noise and so the line that passes through these points will be very sensitive to this noise. Surely, you might think, it is better to randomly choose 3 points (or maybe 4?) since you could get a better fit.

The problem with this intuition is that the more points you sample to fit a line, the greater the likelihood that one of the sample points will be an outlier (and if *that* happens then your fit will be garbage). That means that you are going to have to sample more times in order to find pairs of points that are both inliers.

Let  $w$  be the probability that if that a randomly chosen point is an inlier, so  $0 < w < 1$ . Suppose we need  $n$  points to fit a model. In the case of a line  $n = 2$ . Drawing  $n$  points is called a *trial*.

The probability of all  $n$  points in a trial being inliers is  $w^n$  so the probability that at least one of the  $n$  points in a trial is an outlier is  $p_o = 1 - w^n$ .

RANSAC has a number of parameters that need to be chosen. There are thresholds. You also need to decide when a point is close enough to a model that you consider it part of the consensus set, and how big the consensus set should be in order for you to consider the model. These parameters are related to each other, which makes matters a bit trickier.

Suppose you have a threshold  $\tau$  for deciding whether a point is sufficiently near a model, and you would like to estimate  $p$  which is the percentage of points that come within a distance  $\tau$  from the “true” model. For each trial, if all  $n$  points are inliers, then when you fit a model to these points, your model should be close to the true model. In turn, of the remaining  $N - n$  points, roughly  $p(N - n)$  should be inliers and thus the consensus set should be roughly  $p(N - n)$ . I say “roughly” because the fitted model is based on noisy data.

On the other hand, if one or more of the  $n$  points is an outlier, then the fitted model will be junk (e.g. consider the case  $n = 2$ ) and the consensus set will be very small, in particular, much smaller than  $pN$ . So, after carrying out many trials, we will see that most trials produce a small consensus set, but that some trials produce a large consensus set. The percentage of trials that have a large consensus set will be roughly the percentage of trials in which all samples belong to the model. But the latter is  $p^n$ , so this gives us an estimate for  $p$ .

## Estimating a vanishing point

Suppose you have run a Canny edge detector or some other edge detector and so you have a set of image points and orientations  $(x, y, \theta)$  where  $\theta$  is the direction of the gradient of the image intensity, i.e. perpendicular to the edge. Each such triplet defines a line

$$(x - x_v, y - y_v) \cdot (\cos \theta, \sin \theta) = 0$$

where  $(x_v, y_v)$  is the vanishing point.

Estimating the location of the vanishing point requires estimating the intersection of such lines. This problem would be trivial to solve except that: (1) only some subset of the lines in the 3D scene will be parallel to each other and so we don’t know which image edges to use, and (2) the estimated values of  $(x, y, \theta)$  typically are noisy.

Notice that this problem resembles the problem we discussed above of fitting a line to a set of points. There, (1) we talked about inliers and outliers, namely points that belonged to a line and those that did not, and (2) the positions of the points that belonged to a line were noisy. I argued that when the percentage of outliers was non-negligible, a least squares fit on all the data didn’t make much sense and suggested instead that we use a method such as the Hough transform or RANSAC. The same idea holds for estimating vanishing points.

**“Hough transform” approach 1**

First assume that the vanishing point lies within the limited field of view of the image. (This is a very strong assumption and in general we don't want to make it. But it is a good place to start.) Given  $(x_i, y_i, \theta_i)$ , our model for a line through  $(x_i, y_i)$  and perpendicular to  $(\cos \theta_i, \sin \theta_i)$  is

$$(x - x_i, y - y_i) \cdot (\cos \theta, \sin \theta) = 0 .$$

Here is a sketch of an algorithm we could try:

```

for each edge element (xi, yi, theta_i){
  if | cos(theta_i) | < 1/sqrt(2)
    for x in 1:Nx
      y = round( (yi sin (theta) + (xi - x) cos(theta)) / sin( theta) )
      if (1 <= y <= Ny)
        count(x,y)++
      endif
    endfor
  else
    for y in 1:Ny
      x = round( (xi cos (theta) + (yi - y) sin(theta)) / cos( theta) )
      if (1 <= x <= Nx)
        count(x,y)++
      endif
    endfor
  }
  find (x,y) that maximizes count(x,y) // there may be several peaks

```

This algorithm will be very sensitive to noise, and so to make it work you would need to do more. For example, to account for noise in the  $\theta_i$  estimate, you could add a loop over a small range of angles centered at  $\theta_i$ . You could also weight the vote by the inverse distance from  $(x_i, y_i)$ , since the errors in  $\theta_i$  would lead to bigger errors in the location of the line as you move away from  $(x_i, y_i)$ .

**“Hough transform” approach 2**

A second approach is to take pairs of edges  $(x_i, y_i, \theta_i)$  and  $(x_j, y_j, \theta_j)$ , compute the intersection of their lines, and then vote directly on the intersection points.

```

for each pair of edge elements (x,y,theta) and (x',y',theta')
  if theta != theta'{
    compute intersection (xv,yv) of lines containing these edges
    count( round(xv), round(yv) )++
  }
  find (xv,yv) that maximizes count(xv,yv)

```

### Parameterizing $(x_v, y_v)$ using a unit sphere

One key limitation with the above methods is that often the vanishing point does not lie within the image domain. The first algorithm explicitly assumed that the vanishing point was within the range of a fixed  $N_x \times N_y$  grid. The second assumed the intersection point is represented by an element in the `count` matrix. The algorithm cannot handle the case that the vanishing point is at infinity.

The classic solution to this problem is to consider an image projection sphere, rather than a projection plane. (For example, the back of the eyeball is roughly a sphere.) Place a unit sphere at the center of projection and parameterize each ray that arrives at the camera center by where the ray intersects the unit sphere. (This unit sphere of directions is sometimes called the *Gaussian sphere*.)

### RANSAC approach

How would you use RANSAC to detect vanishing points. As in Hough 2, take two edges and intersect their lines to get a model  $(x_v, y_v)$ . Then for each of the remaining edges, check the distance from  $(x_v, y_v)$  to the line defined by that edge. If the distance is small enough, then the edge is consistent with that model, etc.

At first glance, you might think that this method avoids the problems of the Hough transform, namely that any (finite) vanishing point can be represented and tested. Note, however, that if the vanishing point is far from the origin, then the distance from the vanishing point to any line that is defined by an edge in the image will most likely be very large. This will tend to (incorrectly) favor vanishing points that lie closer to the optical axis. One could reduce this bias again by using the hemisphere, and considering distances on the hemisphere rather than on the projection plane.