We have spent several lectures on how to process images and detect features such as edges and blobs, and estimate image transformations such as translations. Throughout the rest of this course, we will use these "image measurements" to estimate the parameters of many 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 will compare three popular techniques. To keep it simple, we will look at a very basic problem which is to fit a line to a set of points in a plane.

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

$$y_i = mx_i + b.$$

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 experimentor (and is not random) and so only the y is random. In this case, one assumes

$$y_i = mx_i + b + n_i.$$

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.

Total least squares

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 the line

$$x \cos\theta + y \sin\theta = r \tag{1}$$

and we want to find the θ and r that minimize the sum of squared distances to the line. (This is called *total least squares*.) Note that this is a more general equation for a line that the one above since it includes the line $x = constant$. Also note that the θ variable is the direction of the normal to the line and r is the distance from the origin to the line.

The distance from any point (x_i, y_i) to the line is:

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

This little fact is not so obvious, so here's a quick proof of why it is true. (Let me know if you can find a simpler one.) The shortest distance d from (x_i, y_i) is in the direction of the normal to the

line, namely $(\cos\theta, \sin\theta)$, and thus the point $(x_i \pm d\cos\theta, y_i \pm d\sin\theta)$ is on the line. (Whether it is $+$ or $-$ depends on whether the point (x_i, y_i) and the origin are on the same or on the opposite sides of the line.) Substituting into (1) gives

$$(x_i \pm d\cos\theta)\cos\theta + (y_i \pm d\sin\theta)\sin\theta = r$$

and so

$$x_i\cos\theta + y_i\sin\theta - r = \pm d$$

And we are done with out little proof.

How do you solve for θ and r ? You 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 θ, 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

$$\sum_i (x_i a + y_i b - r)^2 \tag{2}$$

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. In class, I sketched out how to solve for θ and r using the technique of Lagrange multipliers, namely minimizing

$$\lambda(a^2 + b^2 - 1) + \sum_i (x_i a + y_i b - r)^2.$$

This is the solution commonly given in Forsyth and Ponce's Computer Vision book, for example.

However, when writing up these notes, I noticed that there is a way to do it that is easier and better illustrates what is going on. If you take the derivative of (2) only with respect to r and set it to 0, you get

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

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 (2) 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 is to considering all the lines that pass through $(\bar{x}, \bar{y})$, and see which perpendicular direction θ 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 assumes[1] 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*). Because we are penalizing by the squared distances to the line, the outliers can have a huge penalty and 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. (There are other methods that are popular, such as EM and SVM that are covered in the Machine Learning course. If you are familiar with those methods, you might want to find a 558 Term Paper topic that uses those methods in Computer Vision.)

Hough transform

For each sample point (x_i, y_i) we are penalizing each line (θ, 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* .[2] The simplest such approach is to give zero error up to some margin distance τ and then constant error beyond that. We would then try to find the (θ, 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 *votes* for points within the τ 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
```

Note that for a given (x_i, y_i, θ) the value of r could be either positive or negative. For an inlier with no noise, though, the r has to be positive, i.e. Eq. (1) on p. 1.

[1] This assumption is made only implicitly above. The assumption can be formalized, for example, one can assume that the noise is a Gaussian of a given standard deviation.

[2] A nice example is "The Robust Estimation of Multiple Motions: Parametric and Piecewise-Smooth Flow Fields", by M. Black and P. Anandan 1994

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, θ)) is known as the *Hough Transform*.

The advantage of Hough is that the outliers have very little affect 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, θ) 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.)

One final note: the Hough transform can be defined for models other than lines. However, note that if your model has several parameters (not just two) then there more votes to cast. For example, if you have points in a 3D space and you are trying to model them as a plane, then each 3D point votes for a 2D subspace of the 3D plane space (that is the set of all planes in 3D is a 3D space). Enumerating all the models in this 2D subspace can be expensive.

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 τ_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*. What is the expected number of trials we do until we have a trial whose points are all inliers?

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 is a trial is an outlier is $p_0 = 1 - w^n$. The probability that we get all inliers for the first time on trial k is $p_0^{k-1}(1 - p_0)$ since this means that we have at least one outlier in each of the first $k - 1$ trials and then all inliers on the kth trial. Note that this probability strictly decreases with k.

The expected value of the trial number in which we first find all inliers is thus:

$$\sum_{i=1} i(1-p_0)p_0^{i-1}$$

which is $1/(1-p_0) = w^{-n}$. In case you are interested, you can prove this as follows:

$$\begin{aligned}
\sum_{i=1} i(1-p_0)p_0^{i-1} &= (1-p_0)\sum_i \frac{d}{dp_0}p_0^i \\
&= (1-p_0)\frac{d}{dp_0}\sum_{i=0}^{\infty} p_o^i \\
&= (1-p_0)\frac{d}{dp_0}(\frac{1}{1-p_0}) \\
&= (1-p_0)\frac{1}{(1-p_0)^2} \\
&= \frac{1}{1-p_0}.
\end{aligned}$$

So, for example, if you are trying to fit a large number of 2D points with a line model and only 20% of the points are inliers, then the expected number of trials you need to pick two inliers is $\frac{1}{(\frac{1}{5})^2} = 25$. So, if you were many more trials than that (say 100), the probability would be very high that at least one of your trials would contain only inliers.

For more details on RANSAC, see `http://cmp.felk.cvut.cz/ransac-cvpr2006` and also see `http://en.wikipedia.org/wiki/RANSAC`, which has links to classic papers and Matlab toolboxes.