Chapter 15

Lecture 15: Homographies, RANSAC, and panoramas

Wednesday, March 30, 2011 MIT EECS course 6.869, Bill Freeman and Antonio Torralba
Slide numbers refer to the file "15RANSAC2011.key"

slide 3 In the general case, when we photograph a scene from two different cameras, a given set of points in one image (for example, from camera A, as shown here) can match to anywhere on a set of lines in the other image (camera B, shown at the right). As you recall, these are called the epipolar lines, and the position along the epipolar line of the matching point in the second image depends on the 3-d depth of that point.

slides 4-6 It turns out that there are at least two conditions under which the mapping of the points from camera A to their positions in camera B is a deterministic function, independent of the 3d depths of the depicted points. Do you know what those two conditions are? (Dramatic pause.) One is camera rotation about the focal point. We'll examine that now. But first, what is the second condition when the mapping that depends only on the 2d positions of the mapped points? When what is being imaged lies in a plane, then one can show (we'll show) that the mapping to the second image becomes a simple function of the 2d position of each point.

slide 7 Let's examine the case of camera rotation first. To remind us, a pinhole camera projection of a scene to an image looks like this. (Of course, physically the plane of projection is on the other side of the focal point relative to the scene, and then gives an inverted image. But it's often easier to just put the projection plane in front of the pinhole, and remember that the real image is inverted from that.)

slide 8 Now, just to get an intuition, let's look at a top view of this, in a flat-land, 2-d world. The light rays come in from the world to the observer whose camera is its focal point (the pinhole) at this purple spot. We represent the focal plane of any camera, say camera A, by a line in front of the pin hole. (of course, it's really a plane, but we're drawing it as a line here for simplicity). The intersection of the incoming light rays with that focal plane tells where that ray is imaged on the focal plane of camera A.

Any camera that shares the same pinhole (or focal point) position with camera A images the incoming rays on to some other focal plane, say this here for camera B. The rendered positions of the light rays we saw from camera A's projection are simply the intersection points of those same rays with the focal plane of camera B. That projection point will be the same no matter what the depth is of the surface reflection that generates this light ray.

slide 9 Just to confirm that, suppose we keep everything the same, but move the pinhole projection position of camera B down a bit in the slide. Then you can see that the position of a rendered ray from camera A now depends on the position that that ray came from in the world.

slides 10, 11 To reiterate, if we rotate the camera, we generate a 1:1 mapping in the 2-d plane relating the location of feature points in the two different images. Now the question is: what is that mapping? If you naively try a translation, you'll find that, in general, you can't find a relative position that lets all the points overlap, ie, that doesn't create a seam if you lay one image on top of the other. Here, with this image pair on the top, related by a camera rotation, if we try to put the left image on top, and position the images as well as possible, we get the seam here. If we try to put the right image on top, we get a seam, too.

slides 12-14 So let's look at the math and see what we expect the relationship to be. The relationship will turn out to be what we call a "homography", so we title the slide that way.

Without loss of generality, let's put the origin of our coordinate system at the pinhole, or nodal point, of our two cameras, indexed 1 and 0, related by a rotation about the pinhole. Using homogeneous coordinates, let's see where a 3d point, (a 4-vector), $\vec{P}$, is rendered in each of our cameras, then see how rendered points relate to each other.

Let $\mathbf{A}_0$ and $\mathbf{A}_1$ be the perspective projection matrices that render $\vec{P}$ to camera focal plane positions $\vec{x}_0$ and $\vec{x}_1$, respectively. So $\vec{x}_0 = \mathbf{A}_0\vec{P}$ and $\vec{x}_1 = \mathbf{A}_1\vec{P}$. Without loss of generality, define the coordinate system to be such that

$$\mathbf{A}_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} \\ 0 & 0 & 0 \end{pmatrix} \tag{15.1}$$

Because camera 1 relates to camera 0 by a rotation about the coordinate origin, if we rotate the 3d world, instead of the camera, we will get the same result as switching from camera 0 to camera 1. ie, to find the projection matrix for camera 1, we can rotate the point $\vec{P}$ opposite to the camera rotation, then use the projection matrix for camera 0. (If there was any translation component between the cameras, that procedure wouldn't work) So we must have

$$\mathbf{A}_1 = \mathbf{A}_0 \begin{pmatrix} - & - & - & 0 \\ - & \mathbf{R}_3 & - & 0 \\ - & - & - & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{15.2}$$

$$= \mathbf{A}_0\mathbf{R} \tag{15.3}$$

Let's first look for a linear relationship between the projections of the point $\vec{P}$ in the two cameras. Let's look for a 3x3 matrix $\mathbf{M}_{10}$ such that $\vec{x}_0 = \mathbf{M}_{10}\vec{x}_1$. inserting the projection matrices, we have

$$\mathbf{A}_0\vec{P} = \mathbf{M}_{10}\mathbf{A}_0\mathbf{R}\vec{P}. \tag{15.4}$$

This must hold for any vector $\vec{P}$, so we must have

$$\mathbf{A}_0 = \mathbf{M}_{10}\mathbf{A}_0\mathbf{R}. \tag{15.5}$$

If we multiply from the right by $\mathbf{R}^{-1}$ and then by

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f \\ 0 & 0 & 0 \end{pmatrix}, \tag{15.6}$$

we have (because $\mathbf{A}_0\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$)

$$\mathbf{M}_{10} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{pmatrix} \mathbf{R}^{-1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} \\ 0 & 0 & 0 \end{pmatrix}. \tag{15.7}$$

slide 15 Thus we see that the relationship between the 2-d image locations of corresponding points between cameras rotated about the pinhole projection is a 3x3 matrix, in projective coordinates. This relationship is called a homography. We find the corresponding point by multiplying $\begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}$. by the 3x3 matrix $\mathbf{M}_{10}$, then dividing the first two coordinates of the result by the third coordinate in order to find x_1 and y_1.

slide 16

Now let's return to the second way we have a functional relationship between the coordinates of points in one camera and the coordinates of points in another camera: when the viewed scene lines in a plane. Szeliski derives in his book the fact that images of planes are related by homographies: See Szeliski book, section 2.1.5, "Mapping from one camera to another".

slide 17 Knowing the homography, you can rewarp images to be as they would appear if your focal plane were parallel to the planar surface. A homography is not an affine warping–it's a non-linear spatial mapping, in which the image coordinates, when described in homogeneous coordinates, are related by a matrix transformation.

slides 18-20 Suppose we click on 4 points that we know should be in a square. Here's how we can solve for the homography that tells us how to warp one image into the other. We know that in homogeneous coordinates, the positions in each image satisfy this relationship, where we want to solve for the unknown 8 parameters of the homography matrix (there is an arbitrary scale factor, otherwise it would be 9 parameters to solve for).

Remembering that to recover the Euclidean coordinates, we must divide by the 3rd coordinate, we have, for the ith pair of corresponding points in the two images,

$$x_i^{'} = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \tag{15.8}$$

$$y_i^{'} = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \tag{15.9}$$

We can then re-write that, pulling out the h_{ij} matrix coefficients that we seek to solve for,

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i x_i^{'} & -y_i x_i^{'} & -x_i^{'} \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i y_i^{'} & -y_i y_i^{'} & -y_i^{'} \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{15.10}$$

Stacking these all up for each point i, with n points creating a 2n by 9 matrix, we have the matrix equation,

$$\mathbf{A}\vec{h} = \vec{0} \tag{15.11}$$

We want to solve that equation in a least squares sense, ie, minimize $|\mathbf{A}\vec{h}^2|$, yet we want to avoid the trivial solution that $\vec{h} = \vec{0}$. Because the homography matrix is only defined up to a normalization constant, let's constrain the length of $\vec{h}$ to be 1. Then (as we did with camera calibration) the unit vector $\vec{h}$ that minimizes $\vec{h}^t\mathbf{A}^t\mathbf{A}\vec{h}$ is the eigenvector of $\mathbf{A}^t\mathbf{A}$ with the smallest eigenvalue. You'll need 4 or more correspondences (8 rows of this matrix relationship) to solve for the unknown parameters of $\vec{h}$.

slide 21 Here are some image rewarpings made using that method.

slides 22, 23 This slide summarizes how to use this method to make a panorama mosaic. The key step of this recipe that's missing so far is how to robustly compute the homography, we which we'll now address.

slides 24 Here's the task: we have two images, with many feature keypoints detected in each one. Each feature point in each image has a local image descriptor associated with it. We want to find the matching features across each image.

Let's pretend the little squares in the bottom row is a depiction of the 128 dimensional image descriptor associated with each feature, in some ordering of the features in each image. Of course, the orderings don't correspond, and we want to match up the descriptors, and identify descriptors that don't match up.

Once we've done that, we can compute the homography relating the two images.

slide 25 We have two ways to add robustness: when working with individual features, and when computing the homography.

slide 26 Let's not take whatever feature happens to be the closest descriptor to any given image feature. Let's only consider a good match those where the descriptor matches below some threshold minimum distance.

The problem: There's no real good value to set that squared error threshold to so that it will separate the correct feature matches from the incorrect ones.

slide 27 Here's an improvement on that, suggested by David Lowe: He proposed a better discriminator : by examining the ratio between the distance to the first nearest neighbor match, and that to the second nearest neighbor match, he developed a much more discriminative algorithm for finding a reliable feature match. You can see from this plot that if you set a threshold of, say, 0.5 on that ratio, you capture a very high percentage of good matches.

slide 28 Does this solve the problem then? No, not yet. We still have too many outliers to reliably find the homography relating two images. So let's move on to using many features to find the best homography relating two images.

slide 29 Here's the idea, illustrated for simplicity for the global operation of translation (instead of a more general homography). Here are some candidate matches (found, say, using the robust method for individual matches described above). Most are good, and consistent with each other, but some are clearly outliers–bad matches.

slide 30 We'll use a very useful algorithm, called RANSAC, for robustly finding global models for many candidate elements, many of which can be corrupted by noise. This algorithm is used in many different vision contexts, not just in making panoramas. The main idea is to select a subset of the elements to be fit, find the model implied by that subset, then to count the inliers.

slides 31-34 So we pick one candidate model, and count the number of inliers that fit with this model. Here, for this first model, there are zero inliers.

Then, let's try a different model. Now, we have 4 inliers. To compute our final model, we average the models implied by all the inliers.

slide 35 This is formalized in a classic paper introducing random sampling with concensus, RANSAC. We'll describe the algorithm in the context of estimating a homography between two images, but it's easy to see from that how to generalize to many different model-fitting tasks in computer vision, where many different elements can contribute to estimating a global model.

slide 36 Here's the basic idea, illustrated for a homography. In the RANSAC loop:

- Select four feature pairs (at random)
- Compute homography H (exact)
- Compute inliers where $|\pi^{'}, H\pi < \epsilon|$
- Keep largest set of inliers
- Re-compute least-squares H estimate using all of the inliers

slides 37, 38 Let's examine an example, line fitting. To find a candidate homography model (8 numbers), we had to select 4 matching pairs of points. Here, we just need to select two points to fit our line. So from all the points, we randomly select two. We find the model those elements imply, then count how many other elements fit that model. We have previously picked a distance-from-line threshold, so we determine the number of inliers by counting how many other points are within that distance threshold from the line. In this case, there is one other point, which gives, including the sampled points, 3 inlier elements for this model.

slide 39 We repeat the process for other random draws of points. These two points (in purple) give a total of 4 inliers, so that model accounts for more of the observed data.

slide 40 These two points give us 9 inliers.

slide 41 These randomly drawn points give 8 inliers. etcetera.

slide 42 After some number of random draws (we'll discuss how many in just a bit), we stop the process and go back to the best fitting model, the one with the most inliers.

Now that we've identified a good model, and a large collection of data points that follow that model, it would seem quite natural to do one last model fit, using all the inlier points, not just the two we had used to find all these inliers. That last least squares fit, using all the inlier points, gives a final refinement of the model.

With RANSAC, we flail around randomly, looking for a model that explains as many of the observed data as possible. Then we use all those good data points to make the final fit to the model.

slide 43

Let's return to this image pair. Remember we have two stages of feature point match rejection. In the first stage, we just look at the features themselves, and check if the local image descriptor has a sufficiently close descriptor match in the other image, with a sufficiently far away second-best descriptor match. The feature points that fail that test are marked in red here.

Then we run RANSAC and look for the largest set of the remaining feature point matches that satisfy the same 8-parameter homography. The feature points that passed the first test, but failed the 2nd test, are marked in blue here. Note many of those blue points are features that just happened to find a good match in the other image (the two sides of this atrium look very similar!), but which aren't two views of the same physical point. The remaining points, which all satisfy the same 8-parameter model, are shown in yellow and all properly lie in the small region of overlap between the two views.

The found homography lets us warp and merge the two images into a two-image mosaic, shown at the bottom right.

slide 44

Now let's return to the question of how many times to repeat the random draw portion of the RANSAC procedure. Suppose the fraction of our datapoints that are good is G. By "good", we mean within our distance-from-model threshold of their value predicted by the true model. Let's say that if we pick any subset of those inliers, we can estimate a model close enough to the true model to let us find a good set of inliers and thus fit a good model. So the question becomes, how many random draws of point sets do we need to pick before we pick a set composed entirely of inliers?

Let's say our model needs P elements to be drawn. For example, P is four feature-pairs for finding a homography, and two points for estimating a line. The probability that any one element is an inlier is G; the probability that all P points from a model-fitting draw are inliers is G^P.

If we ever get lucky and sample all inliers in a model-fitting draw, then we've solved the problem, because the resulting model will have the most inliers within the threshold and we can identify that we have a good model. There are many combinations of good draws that will lead to a successful RANSAC conclusion. It is simpler to compute the probability of failure, because there is only one way it can happen: we need to draw *no* set of all-inliers in any model-fitting draw. Let's calculate the probability of picking a set with at least one outlier N times in a row. That probability is one minus the probability of drawing a set with no outliers, or $1 - G^P$. For RANSAC to fail, we need to do that N times in a row, which has a probability $(1 - G^P)^N$.

slide 45 Let's look at an example. Suppose only half the feature point matches are good. To estimate the correct homography from a random draw of 4 feature point pairs has a chance of $0.5^4 = 0.0625$, a 6% chance. How many times do we need to do a model-fitting draw before, by chance, we get four points from which we can fit the right model?

With only 100 draws, there is a $(1 - 0.5^4)^{100} = 0.00157$ chance that we *wouldn't* make a draw that would lead us to the right model, or a 99.843% chance of finding the correct model.

If that chance of success isn't high enough for you, you can go to 1000 iterations, which leads to a 1 in 10^{28} chance of failure.

slides 46, 47 But, looking at our image pair, with a small overlap between the images, we may think that a 0.5 inlier fraction is too optimistic. Let's examine two other inlier probabilities, 0.3 and 0.1. With 0.3 inlier probability, we have less than a 1% chance of pulling an all-inlier foursome of feature pairs. Yet with 100 RANSAC iterations, we have a better than 50% chance of finding the right model (the probability of failure is $(1 - 0.3^4)^{100} = 0.44$), and with 1000 iterations, the chance of success is 99.97% (probability of failure is $\frac{1}{3400}$).

But suppose the overlap area between the images is really quite small, and the inlier probability is only G=0.1. Then the probability of any given draw of feature pairs having four inliers is only $\frac{1}{10000}$, but the probability of success still rises sharply as we increase the number of model-fitting iterations, N:

- for N = 100, the probability of RANSAC success is $1 - (1 - 0.1^4)^{100} = 1\%$
- for N = 1000, there is a 10% chance of finding the right model.
- for N = 10000, the probability of success is 64%.
- for N = 100000, the probability of success is 99.995%.

slide 48

In the formula for the chance of RANSAC failing to find the right model ($(1 - G^P)^N$), the number of model parameters, P, enters exponentially in a bad way. The fraction of inliers, G, controls the base of that exponential. The number of RANSAC iterations, N, also enters exponentially, but favorably for finding the right model.

slide 49 Slide 49 summarizes the RANSAC algorithm.

slides 50-60 Brown and Lowe published a nice pair of papers on automatic panorama finding, using SIFT features and RANSAC as we have described in class.

slide 61 They have recently developed an iphone app for automatic panaroma stitching. (That's a nice progression, from research paper to iphone app). I'll show an in-classroom demo of using that application.

slide 62

One artifact you notice in the iphone application is occasional ghosting of some edges and lines, caused by imperfect matches between the images. In this journal paper, Brown and Lowe point out that you can remove those ghosting artifacts by using multi-resolution image blending, as described by Burt and Adelson in their Transactions on Graphics paper in 1983. Each image contributes a weight to the final panorama as a function of the distance from the center of each individual photograph. For each pixel of the final composite, an ownership weight is computed for each image, and the image with the largest ownership component for any given output pixel is computed. These binary masks are then blurred in accordance with the Laplacian pyramid level for the level of detail used in the composite, as proposed in the Burt and Adelson Laplacian pyramid image blending technique. This removes the detail ghosting artifacts, as shown in this figure.

slides 63, 64 A remarkable Siggraph paper from 5 years puts together everything we've been talking about regarding features, camera calibration, and robust feature matching into a beautiful project called

Photo Tourism. Many tourist sites are photographed by thousands of people each year. These photos are uploaded to photosharing web sites, and offer a comprehensive set of images of that location, from different viewpoints, lighting conditions, and positions. If these feature matchers really are robust to viewpoint and lighting, they should work under even these conditions.

slide 65 It turns out they do. Here is a video giving an overview of the system.

slides 66-68 This project uses conventional computer vision techniques, that we've discussed in class, with a novel photo browsing interface.

These are the elements of the scene reconstruction: feature detection, pairwise feature matching, and estimating correspondences between cameras, followed by an incremental structure from motion algorithm. We've discussed all those in class, except for structure from motion (although that's related to stereo shape reconstruction).

slides 69-71 The feature detection uses SIFT keypoints and image descriptors.

slides 72-73 The feature matching uses RANSAC. The model that's inferred between images is not a homography, since we can't assume the images are planar or that the cameras make only rotations from the same position. Rather, they infer the fundamental matrix relating points between the two cameras. (This matrix is related to the essential matrix we discussed in class, but is for cameras for which the geometrical calibration is not known, as is the case for our images downloaded from the web. See also the fundamental matrix song,

`http://danielwedge.com/fmatrix/`

slides 74 The structure from motion component infers the depth for every point and an approximate camera calibration for each camera.

slides 75

Here are some links to code and other summaries of the project.