[SQUEAKING]

[RUSTLING]

[CLICKING]

PROFESSOR: So we're talking about relative orientation, one of the second of four problems in photogrammetry relevant to binocular stereo, as well as motion vision structure, structure from motion. And we developed a solution. And we're wondering about circumstances where that won't work.

And in particular, are there surfaces where we can't determine the relative orientation? And we got there. We found that there were quadric surfaces in that family of surfaces.

Now, keep in mind that this is in a coordinate system that's been specially lined up to make the equation simple. So this is, first of all, the center is at the origin. And the axes are lined up.

So in a more general case, we'd have a more complicated looking equation, where we don't just have the second order terms. But we also have first order terms and zeroth order terms, constants. But in classifying the shapes, this is a convenient form.

And we noted that if we have all positive signs, then we have an ellipsoid. And if A, and B, and C are the same, then it's a sphere. If we have one negative sign, we have a hyperboloid of one sheet. If we have two negative signs, we have a hyperboloid of two sheets.

And if we have three negative signs, then we don't have any real locus. But if we extend it to complex numbers, then we can talk about an imaginary ellipsoid. Now, the particular equation we got from the problem of relative orientation fell in this category.

And actually, the way we can get there is to note that it didn't have a constant term. So it was second order, but it only had the second order term and the first order term. And so think of it as an equation like that.

And that means that R equals 0 is the solution. And so that point is on the surface. And what was that again? Well, that was the origin of the right-hand system. Or in the case of motion vision, that's the place where the camera is at time 2. So that's interesting.

That means that this weird surface is one where we actually have to be on it with our right eye, or move on it, in the motion vision case. Well, it turns out that in addition, if we plug in a minus B for R, that's also a solution. And what is that?

Well, if we move left along the baseline by B, then we're at the origin of the left-hand coordinate system. So actually, that surface has to go through both eyes, so to speak. Or in the case of motion vision, we have to start on the surface and end up back on the surface.

And this shouldn't be surprising, because everything should be symmetrical between left and right. It just so happens that we picked the right-hand coordinate system as our reference. But we would expect the same to be true of the left-hand coordinate system.

OK. In addition, because this is a homogeneous equation, i.e. some polynomial in R is equal to 0 with no constant term, there's no scaling. We can't tell the size of the vector. So R equals kb is also a solution, right? If R equal to B as a solution, or R equal to minus B, then R equal to kb is.

And that means that the whole baseline is in the surface. So this has a number of implications. One of them is, it suggests that, well, maybe this is a rare case. How likely is it that you're on the surface with both camera positions and the whole baseline between them is on the surface?

The other thing it means is that it's a ruled surface. That is, we can draw lines in the surface, which of course, we can't do in the case of the ellipsoid, right? If we draw a tangent, it touches the surface at one place. And then, it goes off into space.

But apparently, the surface we're interested in is ruled. And that means, without doing all the detailed algebra, it has to be that one. Because neither one of these two is ruled.

Also once we've decided that it's a hyperboloid of one sheet, we know that it actually has two rulings. That is, at any given point, we can draw a line that stays in the surface going in one direction. And there's a second one that crosses it.

So that's pretty interesting. And that corresponds to the method used to manufacture tables, chairs, what have you out of straight sticks where you use two sticks that cross. And so it's interesting that we find this surface.

OK. It seems like a very special case. Why are we worrying about it? Well, the thing is that this is the general equation for the quadric. But it also covers a large number of special cases.

Parabolic hyperboloids and whatnot-- you can find the whole list of these special cases online. I won't go through all of them, just focus on one in particular, which is planar. Now, we know that this is the equation of a plane in 3D. And if we take a second equation of some other plane in 3D, now both of these are linear in X, Y and Z.

But if we multiply the two of them, then the product will be equal to 0. And what is that equation describing? Well, it's describing those two planes. And if we multiply it out, we get a quadric.

We get something that has up to second order terms in X, Y and Z. And so curiously, a surface like that falls into this category. And planar surfaces are pretty common in the world. So we need to worry about that.

Now, it turns out that for these to have no constant to them, one of the planes has to pass through the origin of the right system and the origin of the left system. In other words, has to pass through the baseline. And so one of the planes is an epipolar plane.

And what's the image of an epipolar plane? It's a straight line, right? Because it's coming right through your eye. And so you're seeing it edge on.

And so that's not particularly interesting, because we don't see any of its surface. But the other plane is arbitrary. The other plane can be anything. And that's scary, because that means that potentially, when we're looking at a planar surface, we end up with this ambiguity.

And it's not just a problem for binocular stereo and reconstructing topographic maps. But it's a problem for motion vision as well, recovering structure from motion. Because the two problems are really the same, mathematically.

And well, we'll pretty much stop there-- except that we haven't really looked at how the geometry affects things beyond this. So in particular, it seems unlikely that we would run into this situation. Because you have all these special circumstances. But maybe your real surface is pretty close to one of these.

And then, you won't run into this problem exactly. But you have large noise amplification, high noise gain. So we would like to stay away from that situation. And it turns out that the field of view has a big influence on that.

So I won't be proving anything about that. But basically, if we have a large field of view, then this problem becomes much more well-posed, much more stable. If we have only a narrow, small patch on the surface, the chance that that patch happens to be very similar to one of these is high. And so it's pretty unstable.

So high field of view-- a large field of view. And so that led to some amusing things. One of them was that people figured this out pretty early when they started doing aerial photography. But they didn't have the lens technology to build something that had incredibly high quality image reproduction, which they needed so they could see a lot of detail.

Low radial distortions-- so they didn't have the image distorted, and also a large field of view. So what they did was, they stuck a bunch of cameras together into a very rigid structure. So there was a set of steel beams. And you stuck these cameras on there.

And they are called the spider heads, because spiders have eight eyes. And so there's a similarity to this solution. Of course, it doesn't mean now that you have to calibrate these cameras relative to one another so that out of the individual pictures, you can compose a mosaic picture as if it was taken by a single camera with a wide field of view. But it makes it clear that this was a well-known problem going back 100 years.

At least from a practical point of view, the mathematics took a bit later. OK. Relative orientation structure from motion-- that was number 2. Let's go on to number 3.

And as we go along, we do less and less detail, because a lot of the basic ideas are common to all of this. OK, so relative orientation-- let's go on to interior orientation, which is basically camera calibration.

And you'll say, oh, but we did that. Well, we had one idea, which was using vanishing points. So the problem was finding X0, Y0 and F, the principal point and the principal distance.

And we had one way of doing that, which was to image a rectangular brick shape. So our calibration logic could be a brick machined more carefully. But that was the basic idea.

And that works. It's not very accurate. It's hard to make it very accurate.

And so what we want is a more general method. Also we need to take into account radial distortion. And the method that we developed using a rectangular brick doesn't lend itself to that very well. So what's this about radial distortion?

Well, I mentioned that we can make these glass analog computers that are so powerful in redirecting rays into just the right direction so they come to fine focus and give us a very detailed image, but there are trade-offs, all kinds of trade offs. And I mentioned that in fact, it's impossible to make a perfect lens. And you basically have to decide what you're going to put up with, and what not.

And radial distortion is something that generally, unless you are taking a picture of an architectural structure with lots of straight lines, you don't notice much. So if you're taking pictures of people, and forests, and cats to post on YouTube, it doesn't really matter that there's some radial distortion. Because nobody will ever notice it.

And so in designing lenses, a lot of the other problems were reduced in difficulty by allowing radial distortion, which basically just means that there's some point in the image-- the center of distortion. And when we express coordinates in polar coordinates, then the image doesn't appear where it should. But it appears somewhere else along that line.

And that error varies with radius. And it typically is approximated using a polynomial. So the usual notation is something like this.

So you can see that delta x delta y is proportional to x and y. So it is a vector that's parallel to the radius vector. So it is along that radial line. And then, the lowest order term is r squared. And why is that?

Well, I'll leave that for a potential homework problem. Why is it we're not including k0 of r, or something? Then, the next term is after the fourth. And in many cases, the first term is good enough to get you somewhere.

And so often, we only find that first term. Yeah, towards the edge. But those coefficients tend to be very small.

So if you buy a telephoto lens from, say, Zeiss, you get with it a plot of the radial distortion. And they're hand calibrated. That's why you pay a lot of money for those lenses.

And you'll see that there's this quadratic rise. But there's also a higher order drop. And that's usually taken care of by that second term.

So yeah, the distortion gets worse a lot as you go out towards the corners of the image. And it's not much of an issue in the center. In machine vision, often, we can get away just with a first term quadratic, just make it simple.

Now, how do you measure this? Well, a famous method used in the past is the method of plumb lines. So you go into an area with lots of space like a parking garage.

And you take some string and some weights. And you hang the weights along those strings. And why do you do this?

Well, because now, first of all, the strings will be straight lines. They'll be stretched out by the wire. And then, they'll be parallel, because presumably, gravity points in the same direction in those places.

They're not far enough apart. And then, you take a picture of that. And then, your picture might look, if you exaggerated, might look like that.

And so first of all, you can see that there's a distortion. And then also, you can see what type it is. This is called barrel distortion, because it looks like the staves on a barrel.

In some cases, you might find the opposite. And that's called the pincushion distortion. I guess we don't do much sewing by hand anymore in our modern world, anyway. So that may seem like a strange concept. But people used to need a way to put away their pins.

And they had a little cushion. And the shape of the cushion is the shape that we see there. So they're pins on the sine of k1.

And the other thing is, of course, once you've got these images, you can look at the radius of curvature of these lines. And you use them to estimate k1. But we'll do it by actual measurement of images.

One subtle point here is, do we want to go from undistorted coordinates to distorted? In other words, we'll take our perspective projection equation. That tells us where things should appear. And then, we look at the image to see where they actually appear.

And then, we fit this polynomial approximation to that. Or do we want to go from the distorted to the undistorted, again using a polynomial? And why might we want to do that? Well, because the distorted is something we actually can measure.

We don't have a way of measuring the undistorted quantities directly. So the two are related, of course, by what's called the series inversion. Not a surprise that that's what it's called. But it's not something that's usually taught.

And it's not entirely trivial. But you can automate it and use some algorithm that will take you from the polynomial going in one direction to the polynomial going in the other direction. The way it affects us is that it affects what coordinate system we want to do the final optimization in.

Because obviously, it's going to be easier to do the optimization in one of those, depending on which way around you've expressed the polynomial. And I guess we're going to try and minimize the error in the image plane. So from that point of view, it may be that we want to go from undistorted to distorted.

But we'll hang onto that. So okay. By the way, radial distortion-- how about tangential distortion? Well fortunately, we don't have to worry about that anymore.

But just for reference, this is what it would look like. No, sorry. Minus y plus x. Again, polynomial-- something growing with power of the radius in the image.

So it's not a problem in the middle of the image. And then, it gets worse when you get to the corners. And now, it's in a direction that's orthogonal.

So delta x delta y-- that vector in 2D-- is now perpendicular to this vector, to the vector xy, right? So if we take the direct product, you can see here. So that used to be a problem, because imaging systems were electromagnetic vacuum tube apparatus.

And in addition to radial distortion, they would have tangential distortion based on exactly how you placed those electromagnets, and so on. It's not a problem in modern devices, because the chip has perfect geometry. And the lenses, if they're rotationally symmetric, only have radial distortion.

But just be aware of the fact that it's called radial distortion. Because there's another one. There are some additional factors that we're ignoring, because they're small, and because they depend on the quality of the lens assembly.

One is called decentering. And so in particular, if the center of distortion is not the same as your principal point, so you're distorting about a point other than what you would normally consider the center of the image as far as perspective projection is concerned, then you're going to get an offset that depends on position. And usually, it's very small.

Because usually, things are assembled accurately enough for that not to be a problem. But if you want to do really high quality work like aerial photography, then you do need to include that. Another thing to consider is if the image plane is tilted. So here's our lens.

And now, your image-- of course, it won't be tilted that much. But it's a mechanical thing that somebody built. So there's a possibility that there'll be some small error there.

And that, of course, means that the magnification isn't quite constant across. And the focus will also be an issue. But if it's a very small effect, the focus won't be affected much. But it will still introduce a distortion.

And it turns out that these two are related. So if you want to, you can have a more complicated model for distortion. We're not going to do that. But it turns out in the end, when we do the nonlinear optimization, you can have whatever you want.

We're going to start off with some closed form formulas. But at the end, we're just going to throw up our hands and say, oh, this is a difficult nonlinear problem. We'll just give it to one of those packages. And at that point, your model of distortion can be more complicated without high penalty, other than overfitting problems, like you've put in so many parameters that it's going to specifically tune it to your experimental measurements. And the resulting apparent high accuracy is bogus.

But beyond that, we can have more complicated-- so what's the strategy? So what we're going to follow is size camera calibration method with some modifications. Yeah. Well, that's a good question.

Because we find that mechanical adjustments and fine tuning at the manufacturer is expensive. Software solutions tend to be cheap. And so in modern times, it's been mostly a matter of extending the model of distortion and having a couple more parameters to tune.

In the past, it was indeed a matter of fine tuning when you manufactured. It wasn't done for anything except aerial photography, where you want the geometry to be absolutely straight. And they would have fine adjustments.

And if you tweaked any of them, your warranty was dead. So naturally, they spent a lot of effort to get it squared up. And in some cases, the fix was not actually adjusting the tilt of the image plane, but introducing a prism wedge, a very small angle that would compensate for it.

And you'd measure how much tilt there is in the image plane, and then go to the storeroom and pick out the compensating element that would just get rid of that component. OK. So Tsai came up with this scheme. And it involves, as you might imagine, a calibration object.

And the calibration object could be anything that you know coordinates on very accurately. And we'll have to make a distinction between planar calibration objects, which obviously are easier to make and keep in the storeroom, and make very accurately using lithographic reproduction methods, or three-dimensional calibration objects like the rectangular brick that we talked about, which are harder to make, harder to maintain accurately. And then on the other hand, they have some advantages in terms of calibration.

So there's a tension there. So we get correspondences, this time between image points and known points on this three-dimensional object. Now, what makes it not quite so easy as when we were talking about the vanishing point method is that we're unlikely to be able to determine, by getting a tape measure, what the relationship is between the calibration object and the camera.

So they're sitting on the floor. We take a picture of it, put the camera on a tripod. And we can go out, and we can measure how far is the first point on that object.

But then, how is it rotated in space? And it's just not practical, particularly since you don't actually know where the center of projection is, that front nodal point that we talked about when we're talking about image projection. And so that means that we need to add exterior orientation.

So rather than find just the interior orientation of the camera as we did when we used vanishing points, now we're going to solve the problem of figuring out where the calibration object is in space and how it's rotated, as well as finding the camera parameters. And that produces much more accurate results, because there aren't any external measurement errors, or errors because we don't know, in this complicated lens with many elements, exactly where is the front nodal point.

It's not like there's a little mark on the side. Because there can't be a mark on the side. It's right inside the lens. So how would you denote it?

OK. So let's see. So that makes it more complicated, right? Because interior orientation has three degrees of freedom, if we ignore distortion for the moment.

And how about exterior orientation? Well, that's translation and rotation. The translation and rotation position, the calibration object. So that's six degrees of freedom.

So we've taken something that's pretty simple only, has three unknowns. And we've turned it into something that has nine. But it does actually make the problem simpler and produces much more accurate results. OK.

So in the interior orientation, we have the good old prospective prediction equation. OK. So xc, yc, zc is in camera coordinates. So if we know some point in the camera coordinate system, we can calculate the position of the image.

And x0, y0, f-- that's the interior orientation. Right? It's the principle point and the principle distance. OK.

Now, the strategy here is going to be that we try to eliminate some parameters that we don't like that are difficult to deal with, like radial distortion. So we're going to try and find a method that, right away, modifies the measurements in such a way that the results are not dependent on radial distortion, and then get closed form solution for some of the parameters out of all of these parameters, and then finally, when we no longer can find closed form solutions, resort to number crunching. And so why do we even bother with this?

Well, because the numerical methods minimize some quantity which has multiple minima. And we want the true one. We don't want to get stuck in the wrong local minimum.

And so we need a good initial guess. And this is how we get the good initial guess. So in the process, we're allowed to violate many of the principles that we've established, because this isn't going to be the answer.

This is our first guess for the number crunching iterative solution. And so for example, we said that we should be minimizing the error in image position. But that's very hard to do directly.

So we're going to minimize some other area that is related to the image position error. And that's OK, because we're not going to stop there. This is just to get the initial condition.

OK. So the xIyI-- that could be just the row and column in the image sensor. Or it could be millimeters from some reference point. But it's very convenient to just use the row and column numbers.

And then for f, well, f could be in millimeters. But it could also be in pixels. Suppose that our pixels are square. And we just use the row and column number as coordinates for image position.

Then it's very convenient to express f in pixel size. It's 1,000 pixels. Why? Well, because when we apply the perspective projection equation, then the units above and below match.

And so we can use any units we want-- millimeters or pixels. So OK. Now, so there are three parameters here. And then, we add to that exterior.

And that of course, is rotation and translation. And so we have a vector for the camera, which is going to be a rotated version of vector for scene plus some translation. So this is again, the camera. And so that's, of course, the xc, yc, zc we've talked about over here.

And this is the scene, or object, or world coordinate, or whatever you want to call. It's not really a world coordinate system. It's a coordinate system in the calibration object.

So we know the calibration object very accurately. And we know its coordinates relative to some system that's embedded in that object, like maybe the corner of that cube, or rectangular brick. OK. Now as I said, we're going to ignore our better instincts.

And for a start, we're going to use rotation matrices here. 2, 3r, 2 xsyszs. OK, so this, of course, is a coordinate in the collaboration object in its own coordinate system.

And then, we're going to rotate that. And then, we're moving the object out by this distance. And of course, that's the unknown.

This is unknown. And that is unknown. And as I mentioned before, we're using the equations in a weird way. Because normally, you would use these equations to take a position in the calibration object and transform it into a position in the camera object.

That's what their purpose in life is. But we're turning this upside down. Those are the things we know. And we don't know what this matrix is.

And we don't know what that vector is. And so those are the things we're going to try and recover. OK.

So now, we combine interior orientation and exterior orientation. And what we get is-- and a similar equation for y. r21 xs plus r22 ys plus-- and then, the bottom is exactly the same as up there. OK.

And again, written that way, it basically allows us to map from coordinates in the calibration objects to image coordinates. But we want to use it instead to recover as many of these things as we can. OK.

Now, I mentioned that it'd be convenient to get rid of the problems of radial distortion and fine tune things right at the end to get those coefficients. And also it turns out to be difficult to get f at first, and tz. So f occurs here.

Now, one way we can deal with that is to look only at the direction in the image. So I guess it disappeared up there. But if we work in polar coordinates, radial distortion just changes the length. It doesn't change the angle.

Similarly, if I change the principle distance f, all that happens is that the image gets magnified with f. And so it moves radially. And the same happens if I change the distance z to the object. All that happens is the magnification changes.

And so again, we're moving along the radius. So the idea is, let's do something to deal with that angle and forget the radial distance in this polar coordinate system. And so for example, we can do this. We can divide these two equations, right?

Because now, we've gotten rid of f. We have a new equation that doesn't involve f. And let's see. xs plus-- and we also have gotten rid of tz. 23 zs plus ty. Right?

So tz occurs here and there. And those two terms cancel. And so we're just left with tx and ty. So we've combined two equations, two constraints. We get a new one, which has fewer of the unknowns in it.

So it's going to be easier to find. So now, we cross-multiply. And we gather up terms. And we get xs yi prime r11. And I'm gathering up terms in such a way that it's clear what are the unknowns.

And of course, in our case here, the unknowns are the components of r, and tx and ty. xs-- that's going to be-- well, these two are equal. And if I bring this over to the other side, I get this minus that equals 0.

So there's an equation. And let's look at that equation to see what's in there. So first of all, xs and ys and zs-- that's coordinate and the calibration object. We know those.

Then, we've got xi prime and yi prime. Those are image coordinates. And we've measured those. So the things in parentheses are known. OK, just a second.

And what's unknown are r11, r12, et cetera, et cetera. So this is a linear equation in those unknowns. Oh, OK. They are these things.

So let me just-- yeah. Thanks for pointing that out. So in order to do this, we need to know where the principal point is so we can subtract it out. And of course, we don't actually know where it is.

But for this purpose, we just need an approximation. And so we can take the center of the image sensor. Unless we know better, we can take half the number of rows and half the number of columns, and put it there. Now, that's going to be a problem if we're dealing with image points that are right close to it, because the directions to those points is going to be affected a lot by any small error in our guess at what the principal point is.

So what we do is-- and what Tsai doesn't mention is that we throw away all of the correspondences that are close to the assumed center of the image, right? Because if we're dealing with the direction of a ray out here, and this center is wrong by some small amount, that's not going to have a huge effect on the direction. But if we're dealing with a point in here and we move the center, that's going to have a large effect on its angle.

So we're cheating by saying, oh, we have a guess at what x0 and y0 are-- the principal point, which in fact we're trying to determine. But it's OK, because it's only an approximation. And we don't use the data where it matters, where that effect would be the most severe. So OK.

So we subtract out-- we reference everything to the image center that we assume is close to x0 and y0. And then, we get this linear equation, the unknowns. And we get one of these equations for every correspondence, right?

So every time we say, oh, this point in the image is that point on the calibration object, we can write down one of these equations. And of course, the xs and yI's will change as we go. OK. Now, it's a linear equation.

And how many do we need? Of course, with just one, we can't solve, because we've got a bunch of unknowns. So we need to count how many unknowns we have.

OK. So now, we got rid of some stuff. So what's left? So what's left is-- so those appear in there, as do these.

And tx appears, and ty. So what doesn't appear? Well, so out of all of the unknowns we're trying to solve for, these are the ones that appear in the equation. There are eight.

That seems to make sense. Eight of those. And then, there's a bunch more that we don't. Now, keep in mind that we're not enforcing orthonormality of the matrix, right? Because we're pretending that those are nine unrelated numbers, not three degrees of freedom of rotation.

So there are really six numbers here, when we know that rotation only has three degrees of freedom. Now as for these other three here, if we ever get these six, we should be able to get this one by cross-product, because we know that the rows of the rotation matrix are orthogonal. And so if you find a vector that's orthogonal to row 1 and row 2, then it's going to be parallel to row 3, right?

And so cross-product or the first two rows gives us something that's parallel to the third row. So we can recover this one afterwards. But right now, we're not even enforcing that this is supposed to be a unit vector. This is supposed to be a unit vector.

And these two vectors are supposed to be orthogonal. We're just going to go with this. OK, the eight unknowns-- so that suggests eight equations are needed, eight correspondences. But that's not quite true, because this equation is homogeneous.

The result is equal to 0. And we all learn how to do a linear equations. I don't know about your education, but typically, there's not much emphasis on homogeneous equations. And they come up quite a bit in machine vision.

So it's important to know what to do with them. So one feature of a homogeneous equation is that the linear combination of some variables equals 0. If I double all the variables, it'll still equal 0. So there's a scale factor issue that I cannot, from the homogeneous equation, figure out what the scale factor is.

And well, then a method of solution is, take one of the unknowns. And just set it to whatever you want. So for example, here we might say ty equals 1, right? Why?

Well, because whatever the solution is, I can scale it so that ty equals 1. And conversely, if I get a solution with ty equals 1, then I have some multiple of the true solution. And the equation doesn't tell me what multiple.

I can't figure that out from that equation. So this is a way to proceed. OK.

So that then reduces it to linear equations in seven unknowns, right? Because I've fixed one of them. And so that means that I need seven correspondences.

So your calibration object should have seven points that are easily identified, preferably from any point of view. And that means you probably need more than seven, because some of them will be hidden. So let's see.

So if you have a cube, you typically hide from a general position one of the eight sides. You'd be left with seven. Oh, that's a nice match.

So a cube is not a bad calibration object. OK. So then out of this, we get some multiple of the true solution. We get r11 prime, r12 prime, and so on.

So this method of solving the homogeneous equations will give us that. Oh, so by the way, if we have exactly seven correspondences, we're going to end up with seven linear equations and seven unknowns. And we know how to solve those using-- I don't know-- Gaussian elimination, or MATLAB, or whatever you want.

What happens if we have more than seven correspondences? First of all, that's desirable. The more correspondences you have, the tighter your solution, the smaller the error is. And with seven correspondences, you will get a perfect fit.

Does it mean you have no error? No. But if you take more correspondences, you can estimate the error.

So it's not only that you get a better answer, but you also get an estimate of what's wrong with it. So typically, you'd use more than seven. And that means that your system of linear equations is overdetermined.

And then, you use least squares, pseudo inverse, standard stuff to find the best solution. OK. But whether it's seven or more, now we have this. And we have to figure out what the actual solution is.

Well, we know that these are supposed to be unit vectors. So we can calculate a scale factor. So we can compute a scale factor to make those be unit vectors.

And oh, what if those two don't agree? Well, that's a good sanity check. If you do this, and you find that those two scale factors aren't approximately the same, then there's something seriously wrong.

For example, you have misidentified the correspondences. You thought this was the point on the corner of the cube on the left, but it wasn't. Then, these will come out different, typically.

In practice, they won't ever be exactly the same. So you can take the average, if you like. And so now, we can scale this vector to turn it into this one. OK.

So what we've done now is, we have a first estimate of everything except f and tz and radial distortion. And this was closed form. If we do pseudo inverse, that's a closed form solution.

OK. So next step is going to be finding f and tz. But while we're here, we're trying to make these unit vectors like they're supposed to. But we haven't really enforced that they're orthogonal. So that'll be another check.

So you could take r11 prime, r21 prime, et cetera, r22 prime. So this is supposed to be equal to 0. And again, if it's not, then that's a potential problem. In practice, it'll never be exactly 0.

But if it's large, then that means, again, something went wrong in your calculation. But we're going to need the full rotational matrix in a moment. And so that means we're going to take the first two row and take their cross-product.

But if these aren't orthogonal, then we're going to get some sort of messy matrix that's not orthonormal, and so on. So squaring up-- so we have two vectors. And they're approximately orthogonal.

How do we get a pair of vectors that is orthogonal, and that's as close as possible to the two vectors we start-- so what is the nearest set of orthogonal vectors? So let's throw it this way.

Here are two vectors, those first two rows of the rotation matrix. And they're not quite orthogonal. And now, we want to make some small adjustment so that we get new vectors that orthogonal. And then, we take their cross-product.

And we have the complete rotation matrix. By the way, heaven forbid that we end up with a reflection matrix. That's something we want to worry about. Now, if we are the ones taking the cross-product, we can make sure that it's a rotation, not a reflection.

OK. Well, it turns out-- and this is boring least squares-- that the smallest adjustment is the following. That is, the adjustment in a is in the direction b. And the adjustment in b is in the direction a.

And so now, how big is k? That's the only remaining question. All right. We want the new vectors to be orthogonal.

And so there's the equation. And we have to solve for k. So you get a dot b plus a dot a plus b dot b into k plus k squared into a dot b equals 0. So there's a quadratic for k. Solve for k and iterate.

Well, if we get the exact right value of k, we don't have to iterate. But we'll see in a second, that's actually not going to happen. Why? Well, look at this quadratic.

The first term and the last term are 0 at the solution, right? We want them to be orthogonal. And so near the solution, those two terms are going to be very small.

And so this is going to be a nasty, numerically unstable quadratic equation. We're not used to that. We're used to seeing more complicated equations being nasty. But this is one case where the quadratic actually fails.

And so instead of solving the quadratic, we do that. Where does that come from? Well, suppose that k is very small already. Then, k squared times a dot b is even smaller.

So forget that. And then, solve the rest for k. And if you need a solution, those two are going to be unit vectors-- a dot a plus b dot b is 2. And so you get minus a dot b over 2.

And that's why I said iterate, because rather than try to solve that quadratic very accurately, you just solve that simple equation and iterate it a couple of times. So instead of using the standard formula for the solutions of a quadratic, we use this approximation. Anyone awake out there? [CHUCKLES]

Is that your standard formula for solutions that are quadratic? OK, probably not. But believe it or not, that is a formula for the solutions of a quadratic.

And it's sad that we don't know this. So what's the other one? Well it's, minus b plus or minus-- right? So this is the one we all taught. It turns out, this is also a formula.

And the way you can check it is that if you have the two roots, x1 x2 product is supposed to be c over a. x1 plus x2 is b minus b over a. So you can easily check that both of these formulas are right by checking the roots product and sum of the roots.

So why do I bring this up? Well, in this formula here, depending on the plus or the minus that you're using, you may be subtracting nearly the same size quantities. And we know that since computers can't represent real numbers exactly, there is going to be a loss of precision.

So if you have 2.111111, and you subtract 2.111 with a 2 in it somewhere, you get a very small number. And you know that you can't really trust that number, because it only has a limited precision. So every time in the case of real solutions, one of the two answers you get is rather poor, right?

Because in one of the two cases, these two have opposite signs. And every time you subtract two floating point numbers, you lose precision. The trick is that the signs over here are the opposite of the ones over here. So you get one of your solutions from this one where the signs match. And then, you get the other one-- I guess I should have written it this way.

And you get the other solution from this one. And this is how you get accurate solutions to quadratics. So little side note there. So we could have used this to get a good answer for k.

But a very simple method is just that iteration. And you can see that k will eventually tend to 0. And then, when you're satisfied that it's small enough for numerical precision on your computer, you can stop. OK.

So that was the tweaking of the rotation matrix components. So now, we have a replacement for r11-- that first row and the second row-- that are actually orthogonal. And then, we can get the third row by cross-multiplying. And we have a full rotation matrix.

Remember, though, that this isn't the final answer. Because we haven't followed our rules about how to do this, how to get accurate results. OK. This is a good time to talk about the planar target case.

So planar targets are very attractive from the point of view of being easy to make, easy to store, and having high accuracy. And so for example, if you had your wheels aligned recently, you might have been at a place that uses machine vision for wheel alignment. And what they do is, they mount a calibration target on the wheel, and then rotate the wheel, or rotate the steering wheel to measure two different axes.

And what's that target look like? Well, it has a pattern on it that has a feature that it's very possible to get incredibly high accurate position of corners of the pattern to 1/100 of a pixel, even better than with our edge finding methods. And how is it mounted? It's planar.

It's mounted on the side of the wheel at an angle. But it's planar. And why are they using planar?

Well, because it's possible to cheaply manufacture incredibly accurate planar patterns. But there's a downside. And so let's talk about that. So here's our planar target.

And I guess we call this coordinate system s. And we can construct a coordinate system there. It makes sense to construct it this way, where x and y are in the plane of the target, and all of your coordinates are known in xs and ys, and zs is 0.

That's the direction perpendicular to the target. Well then, we can follow the same methodology up there. Except now, there are certain terms that don't matter anymore, like r13.

It gets multiplied by zs. R23, r33-- none of those occur anymore. OK. So we get this equation instead, because that term in zs drops out. So big deal.

And now, we cross-multiply. And instead of getting the equation up there, we get the slightly simpler equation. OK. And again, same thing. The things in parentheses are measurements, the things we know.

Then, r11, r12, et cetera are things-- they're the unknowns that we're looking for. But now, there are fewer. So now, if we list the unknowns, there are six instead of eight. And since these are homogeneous equations again, we turn them into inhomogeneous equations by setting one of the parameters equal to 1. And then, we have five equations and five unknowns.

So one great feature of this approach is that we only need five correspondences now instead of seven. And again, usually, we would use more and use least squares to get a more accurate solution. Oh, by the way, what happens if, just by chance, ty in the real world is actually 0, that there's no translation in the y direction?

Well then, this method is going to have a problem. It means that all your other parameters are going to be huge, and probably inaccurate. So that's something about this approach to solving a homogeneous equation. What do you do then?

Well, set tx equal to 1. So I'm presenting it this way. But actually, to get a numerically good solution, you would want to check the result. And if it's the case that ty is actually very close to 0, then switch to the other one.

And so I didn't make a point of that before. But OK. Well before, we recovered the full rotation matrix just by squaring up two of the vectors and then taking a cross-product. And so here, hmm.

Now, we've only got the top 2 by 2 piece of the rotation matrix. And so I won't write down the solution, because that's what you're supposed to do in the homework problem. So OK.

So let's suppose that we can do this either for planar or nonplanar. But you can see how this is different for the planar case clearly. Now, there's another subtlety that I didn't bother with, because it's not really relevant anymore today.

But in the old days, you weren't quite sure about the relationship the aspect ratio of the stepping in the x direction and the stepping in the y direction. Because they were produced by very different effects. So one of them was just lines.

And this is true even of CCD and CMOS sensors, which were pixel-- they were sensors, discrete sensors. But the way they were read out usually was in an analog form. So you took the discrete signal out of your row of sensors, turned it into an analog form. And then, a bot in the computer chopped it up and digitized it, but not in any way related to the size of the pixel step in the row, right?

The frame grabber had its own clock. And so as a result, the spacing horizontally and the spacing vertically were controlled by different things. The spacing vertically was-- I've got different rows in my sensor. I know exactly what that is.

And horizontally, it was, well, what's the relationship between the clock in the frame grab and the clock in the camera? And so we needed another parameter that scaled x relative to y. And it turns out that you couldn't find that parameter with a planar target. And it makes a bit of a mess of the algebra.

So I didn't want to go there because today, of course, you look at the manufacturer's spec sheet. And you know exactly what the aspect ratio is of the stepping in the x and the y direction. But again, it brings out the fact that the planar target is different. OK.

What's left to do? Well, we don't know f. And we don't know tz. And we also don't other things. But let's focus on that.

How do we find them? Well, we use the same equations. And I won't write them out again, just multiply them out.

So this is just a perspective projection equation, where we combine interior with exterior orientation. And you can see that now, we need the full rotation matrix. And it'd be good if it was really orthonormal. And again, the terms in parentheses are the ones that we know at this point.

And so the unknowns, of course, are f and tz. So this is a simpler problem than the one we had before. So this stuff-- all of this, we can calculate, right? Because we've got the image measurements.

And at this point, we've got tx and ty, and the components of the rotation metric. So we can calculate all of that. We can calculate all of this.

And we just need to solve for f. And that means we actually need only one correspondence, right? Because from one correspondence, we get these two equations. And we're looking for two unknowns.

So now, of course in practice, we would never do that. We would use all of the correspondences we can lay our hands on and do least squares. But the minimum number is 1. OK.

So that gives us f and tz. And there is a little problem here, though, which is that I need depth variation. So it's very tempting to do this with your calibration target.

So here's the image plane. Here's your planar calibration target. And here's the lens. All right. This seems like a nice arrangement.

And what's wrong with that? Well, we know that perspective projection has in it multiplication by f and division by z. So if we double f and double z, nothing happens. That's a scale factor ambiguity.

So that means that in this case, you cannot discover f and tz separately. You can only determine the ratio. And that's, of course, unsatisfactory. And so what you need to do is have variations in depth.

And there's issues about how much, and what's the best, and so on. But basically, it's not going to work if it's perpendicular to the optical axis. And this could be-- I don't know-- 45 degrees, 60 degrees, depending on what you're trying to do. And if you go into the place where they do your wheel alignment if they're using machine vision, you'll see that when they mount the calibration target on the wheel-- so they have a camera looking down parallel to the axis of the car-- and they mount this thing on the side of the wheel so it has to stick out. But instead of mounting it perpendicular so that you get the best view in the camera, they're mounted at 45 degrees.

And why? Because exterior orientation is ambiguous if you don't do that. You can only determine the ratio of f over tz. You can't determine f and tz separately. OK.

Now, we're almost done. We've got estimates of most of the parameters. So what's missing?

What's missing is the principal point and the radial distortion. So it turns out, no one's come up with a way of closed form solution for the principal point. And so we just give up at this point and say, okay, now we need to do the nonlinear optimization.

And the idea is that here has an error between xI and xp. OK. So if we have all of these parameters, we can calculate forward from some position in the world to some position in the image. And that means, then, we would hope that that was going to lie right on top of the image point we actually measured.

So the calibration object will have a bunch of points that are easily identified. And then, you look at point number 3. And you pump it through the rotation translation perspective projection.

It gives you a predicted position in the image. And you saw it somewhere else. And that's an error. And that's the thing you're trying to minimize.

And so I can write it this way. That's what I would hope. And of course, in terms of least squares minimization, I would just take the sum of squares of those two terms.

And how do I get the xp and yp? Well, I apply the rotation matrix to translation matrix and the principle point information that I have, as well as the radial distortion. So now, I have something that depends on r, t, x0, y0, f, k1, possibly k2, maybe some more. So I've got a whole bunch of parameters.

And now, I have this huge minimization problem. And as we mentioned last time, there's this wonderful package invented eons ago that does it. And what's it called?

LMdiff. And it's in the MINPACK in the original Fortran, if you like. But it's been translated into lots of other languages. And of course, it's built into MATLAB and whatever.

So now, we just set up this least squares problem where we're trying to come as close as possible to satisfying a bunch of equations of this form, one pair for every correspondence. And we do it by tweaking these parameters. Well, there's this little problem here.

R is highly redundant. It has nine numbers, and only three degrees of freedom. And so we could try and impose the constraint. So this package works for unconstrained minimization.

And so imposing R transpose R equals I, and determinant of R equals 1 plus 1-- that's going to be hard. So what to do instead? Well, what Tsai did was Euler angles.

And what you can do instead is Gibbs vector, which is-- let's see. Omega hat-- so this so this is a nonredundant representation. There are only three numbers. That's the good part.

It has singularities. That's the bad part. This one is nonredundant. It has three numbers.

The bad part is that it blows up if you rotate 380 degrees. Now, if you know that's not going to happen because of the way you set up the calibration object, then that's a perfectly acceptable way of proceeding. That works pretty well.

The other one is, of course, to use unit quaternions, which have no singularity. Right. So we can use that for the parameterization of quaternions. Unfortunately, it's redundant, right?

Because there are four numbers for three degrees of freedom. But what you can do is really simple. You add another equation.

So there's going to be an error term that is proportional to the difference between the size of this quaternion and 1. And you can determine how strongly that's enforced. And as you turn that up, you get the solution.

So that's one implementation that works very well independent of conditions like avoiding 180 degrees. Now, Levenberg-Marquardt finds local extrema. So if you put it down in the wrong place in parameter space, it'll be perfectly happy to walk into some other local minima.

And that's why we had to do all the other work to get an approximate solution first. Otherwise, we could have done away with all of that stuff and just start there. But we can't.

A very important question is noise sensitivity, or what we've been calling noise gain. And we alluded to it in several places, like over here where the calibration plane, if it's perpendicular to the optical axis, then the noise gain on f and tz is huge-- infinite-- while the ratio is perfectly well-determined.

But it's hard to say something general because of this numerical optimization, and because these methods here were approximate. They didn't enforce the conditions directly. So how do you address the noise gain issue? Well, as happens in many cases where all you have is a numerical method-- and this is where the advantage of an analytic method comes to the fore-- if you only have a numerical method, you can use Monte Carlo methods.

So how do you do that? Well, you take your measured image positions. And you add some noise of known statistical properties. You add some Gaussian noise, noise with some known variance or standard deviation. And you do the computation.

And you get a different answer. And you do this many times. And you look at the statistical properties of the answer. And you look at its standard deviation.

And then, the ratio is the noise gain, right? Very straightforward method. Once you've written the code to solve this problem, you just take the inputs and you fiddle with them, and do this many, many times.

And each time you get an answer. And then, you look at the distribution of the answer in the parameter space. And that way, you can do what normally you would do if you had an analytic solution.

And this is how you find out things like-- that this absolutely does not work. If the calibration plane is perpendicular to the optical axis, you get a huge noise gain in a certain direction in parameter space. And you find out that the higher order coefficients of radial distortion past k2 are very poorly determined, that they're very sensitive to any kind of noise measurement. So it's probably not worth trying to get them.

Oh. OK. So what are we going to do next after you come back is go, again, one level up. So we started off real low-level stuff.

Then, we went to this intermediate stuff. The next thing we're going to do is talk about representation of shape, and recognition, and determining the attitude in space. So it's parallel to what we did in 2D.

When we were doing the patterns, we did recognition and attitude determination in 2D. Now, we're going to do it in 3D. And we've got all the tools for it, now, to talk about rotation in 3D. So have a good holiday.