[SQUEAKING]

[RUSTLING]

[CLICKING]

BERTHOLD HORN: We're switching topics. So far, we've focused on sort of early vision, the low level material, what to do with gray levels, how to detect features of certain types, in particular edges. We talked about very accurately finding edges.

We could continue along that line. We could talk about finding so-called interesting points, but let's move on to the next stage where we're actually using that information to achieve some purpose. And so we'll be talking about photogrammetry next.

And the name is composed of two roots, which basically mean measurement and images. So it's about measurement from images. And it has its origins in right after photography was invented because people saw the potential for map making.

You would send someone up in a hot air balloon, which was popular in those days. And then you have a picture, which is kind of a map if the ground is flat and you had the camera just the right way. If you were over hilly terrain, then taking multiple images might allow you to reconstruct a three-dimensional surface.

So that's the kind of thing we'll be looking at. And in particular, we're going to talk about four different problems, which have the names-- and these are just classic problems from photogrammetry, which have been reinvented by people in machine vision and different given different names. But basically, that's what they are.

So this is the one we'll talk about first. And it's about finding the relationship between something in 3D and something else in 3D. And it could be, for example, that we have two different coordinated systems, two different sensors. Imagine the autonomous vehicle with two laser sensors. And we're trying to relate those to coordinate systems.

So they're different kind of tests. One is how do you use the measurements from these two coordinate systems to get 3D information. But before you do that, you first have to figure out how those two systems are related. So you need to find out the coordinate transformation from one sensor coordinate system to the other.

So that's one kind of problem. And the symmetrical problem to that is where we have a single coordinate system, but either they are two objects or the object moves. And we want to know the relationship between before and after. And it's actually just the same problem, and we'll talk about that.

There are some advantages to attacking the 3D problem first. One is that it has a closed-form solution. It's always nice even though, in some sense in machine vision, we often are more concerned with the second problem, which is 2D to 2D, right?

So the main sense that we're concerned with is a camera, and we get images, 2D images. And for example, we might have two cameras, get two 2D images. And we'd like to use them to recover the third dimension. So that's binocular stereo, for example.

That's just one application. And again, there are two types of problems, two types of applications. One is how do I recover the three-dimensional information from those two images. And the other one is how do I first get the two cameras lined up, so that I can do that. How do I find the relationship between these two cameras, where they are in space and how they are rotated, so exterior orientation.

So that it is a mixed problem, 2D to 3D. And this has to do with the situation where we have an image and we also have a 3D model of the thing that we're looking at. And so we're trying to figure out where we are and, again, how we're oriented in space. We're going to keep on hearing about rotation and orientation because that's a problem in all of these areas.

And so I don't know. You could imagine a drone flying over this are, and you have a map of this area. And you're trying to figure out from the image where you are. So that'd be a difficult problem of that form. Or you're on a spacecraft, and you're staring at the stars. And you're trying to figure out the rotation in space of your spacecraft.

And then the last one is very similar. And I just write it the other way around, 3D to 2D. And that is interior orientation. Exterior and interior refers to the camera. So interior orientation is basically your camera calibration.

So we make the camera to certain specs. We have the lenses with certain focal length. And we have some kind of metal case that puts the image in a position relative to the lens, but it's never going to be exactly according to spec.

So in order to make very precise measurements using these cameras, we need to first figure out their properties. And we already talked about finding the principle point, and that's form of interior orientation. So since I have something here, I should have something in the last one.

OK. And in terms of applications, it's kind of obvious. The original application in photogrammetry was making topographic maps. So you'd fly a plane along some predetermined path taking pictures at regular intervals, and then a person would look through an instrument sort of like binoculars. Except you're presenting two of these images, and the person's eye-brain combination lines them up and basically determines the height at any particular point.

And in order to plot the map, there would be an artificial point where you've introduced two points in the two images the person is looking at. And their position determines the height. And so the person basically places that artificial point on the surface they see and moves it around at that height and thereby creating a contour.

And of course, we don't do that anymore. It's all now automated using correlation, then convolution. But before we do that, we need to figure out how the cameras relate to each other.

OK, just to give a really simple example of the kind of thing that happens in photogrammetry-- so humans have many depth cues, a dozen or so at least. And we already talked about some of them. We talked about shading, and we talked about shape from shading.

And another one is binocular stereo. So if you talk to the woman or man in the street and ask them, how do we see 3D, typically they say, oh, we have two eyes. So this is an important depth cue at least from the point of view of how people think we see 3D even if it's not the most important depth cue.

But let's take a look at that and look at it in a highly simplified special case situation. So in this situation, we have two cameras. And we've lined them up so that they are perpendicular to the baseline.

So each of the optical axes is perpendicular to the baseline. And the optical axes are parallel. And the lenses have the same focal length and so on. So in practice, we need to be more careful. And we need to allow for differences between the cameras. And typically, it'll be not very practical to perfectly align them up this way.

There's someone at Draper Lab-- Sutro was his name-- who spent a long time working on stereo for Mars rovers long before there really was a Mars rover. And he was concerned with getting these cameras lined up. And let's just say that he never succeeded. It's too difficult to get them lined up accurately enough.

And so what we're going to do later is actually figure out how these cameras relate to each other. But just to get a feel for this, let's look at this situation. And we'll pick an origin that's halfway between the lenses.

And the baseline length is b, so b over 2 from each of those. And then let's suppose that they're a distance f from their respective image plains, where f is, as we know, not the focal length, but slightly larger. And then we've picked some point out here. And we'll image it in both cameras.

We're leaving out the y direction, which is perpendicular to the blackboard because that's less interesting from this point of view. And so up here, we have a point at X and Z. And our task is, from xl and xr, the position in those two image plans, to determine big X and big Z.

So we have the two images. The object does not image in the same place in the two images. And from the difference, we can calculate where it is, whereas from a monocular image, of course, we have that scale factor ambiguity. We can tell what direction something lies in, but we have no idea how far away it is. Adding the second eye or lens gives us that ability.

Well, we just use similar triangles. So using similar triangles-- because x and y are relative to this origin. And so if we want to have this distance, we need to subtract out half the baseline. And similarly, on the other side, we have to add in half the baseline.

OK, so two equations, two unknowns, we can solve. So the depth is inversely proportional to the disparity. So the disparity is the discrepancy in position if you were to superimpose the left and the right image. And so that gives us directly the depth, the distance.

And that also allows us to determine the sensitivity to error. Obviously, if we're very far away, the disparity will be very small. And then any small error in measuring the disparity is going to produce a large error in depth.

OK, and xr plus xl-- and again, the disparity comes into it and the baseline. And this time, we're sort of taking an average of the positions.

So we can determine the 3D position-- well, we've left out y, but that's pretty easy-- if we have those two measurements. And one thing is clear. If we increase the baseline, the disparity gets larger, and so then the measurement will be more accurate.

If we increase the focal length, then the disparity will be larger, and we improve the measurement accuracy. But of course, you have other constraints on these quantities. You don't want the two cameras to be too far apart if possible.

So for example, in autonomous vehicles, it'd be nice if the two cameras could be close enough together, so you could hide them behind the rear view mirror, rather than have them go all the way across. And if they are that far apart, then you have a calibration problem because they're unlikely to stay perfectly lined up over the lifetime of the car. So you'd have to have some automated way of rediscovering the relationship between those two coordinate systems.

OK, so this is a very special case, not one that one could rely on in practice, as I mentioned. And so we'll have to talk about what happens if these two cameras are not exactly equal and they're not exactly arranged in this geometry. What if there's a rotation between the two cameras? What if the baseline is not perpendicular to the optical axis and so on? How do you calibrate for that? And that's going to be the topic of a later conversation.

So let's dive right in, problem number one. Let's look at absolute orientations. And one version of that is we have two LIDARs, laser range-finding imaging devices, that give us 3D coordinates. And we're going to use that information to get a picture of the world around our autonomous vehicle or whatever.

And to do so, we need to get the relationship between those two systems. I mean, we try to line them up in some reasonable way, but we can't really depend on it. In the case of the topographic reconstruction from aerial photography, it's obviously even worse because the two camera positions aren't connected by some rigid object.

The plane flies along, takes a picture. It flies along, takes a picture. And the pilot, of course, tries to maintain a constant altitude, but the air is not a constant medium. So the orientation of the plane is not going to be maintained exactly. And so we need to compensate for that.

Notice that, in all of this, we're presupposing that we somehow found points. And we've seen how to find edges very accurately, and that's one type of input that we can use. Similarly, there are methods for finding quote, "interesting" points.

And you know, SIFT is an example. And I think I have David Lowe's paper on the stellar site. It's a little opaque. And since he wrote it and patented it, there have been many other variants that typically are computationally much faster, but may not be as accurate and so on. But anyway, we're assuming here that there's some way of finding interesting points and finding them again in another image.

So we're leaving aside the matching problem. We're assuming that's been done. And so, now, we just have the problem of here's a bunch of coordinates in the two systems. What do we do with them?

So I'm going to call these left and right coordinate systems just because one of the big applications here is to binocular stereo. But of course, they could be before and after. So the vehicle took a picture of the world, then it moved, then it took another picture.

So don't assume that, because it's l and r, that that's-- so here's a point in the environment. And we're going to find we'll need more than one point, so let's put a subscript on them. And we measure it in this coordinate system, and we get rli. And we measure it in this, and we get the vector rri.

And again, there's this duality where we could be talking about one object measured in two different coordinate systems, or we could be talking about two objects measured in the same coordinate system. And this is sort of like, is the camera moving or the object moving? It doesn't matter. We just occasionally need to be careful about the sign because the sign will flip.

OK, so now, if we know where those coordinate systems are, we have a very happy situation. We simply project out this ray into 3D and this ray in 3D, find out where they intersect, and we know pi is, that point i. And that's sort of ultimately what we want to do.

And they typically won't intersect. So we'll find the point where they have the closest approach, which is where the line connecting them is as short as possible. And that will be a line that is perpendicular to both of them. So there's an interesting little geometry problem there, how to do that.

But right now, we're focusing on another problem, which is, before I can project these out into the world to find their intersection, I need to know how these two coordinate systems relate to one another. And so how do three-dimensional coordinate systems relate to each other? Well, there's rotation and there's translation. And so our job is going to be to find the rotation and to find the translation.

And we've talked about this before. In 3D, there are 3 degrees of freedom to rotation, and there are 3 degrees of freedom to translation. And that's slightly surprising. You know I'd say, oh, yeah, because it's 3D. So it's got to be three.

But as we mentioned in 2D, rotation only has 1 degree of freedom, not 2. And in 4D, it has 6. So it's n times n minus 1 over 2, not n. And so it's a fortuitous accident that, in 3D, 3 times 2 of over 2 is 3.

So it's slightly confusing. It makes it look like rotation is simple when it isn't. OK, so we're looking for six numbers. And we will need enough constraint so that we can find them.

So we're looking for a transformation of this form where this is a translation which is fixed. That is this formula applies to all the points, Pi and Pj. And this is our rotation. And I've purposefully written it this way with a parenthesis to indicate that we are not going to insist on that being represented as an orthogonal matrix, which is what would happen if I left out the parentheses. You would assume that I'm talking about a 3 by 3 matrix multiplied by a vector, but we'll be looking at other representation. And we'll see why.

OK, so those are the unknowns. So with an equation like this, there are typically two ways of using them. The one is you know the rotation and translation. You plunk in r of l, and you compute r of r. That's sort of the easy thing and certainly what we use it for, but that's easy.

Our problem is different. We know some rl, rr pairs. That is there are certain points we've measured in both coordinate systems. And our job is to find the rotation and the translation.

OK, now, before we go on, we should talk about some properties of rotation. And we already mentioned them, but let me do it again. So if we represent r as a 3 by 3 matrix, it's better be orthonormal. And that means that the dot product of rows is 0 and that the dot product of rows with themselves is 1.

And so a quick way of writing that is that way. That is the matrix times its transpose is identity. And you can check that. That means that, I guess, the columns, if we treat the columns as vectors, those are orthogonal.

And they're unit vectors. And it turns out that you can prove that, if the columns have that property, then the rows have that property. So we could also write it as r times r transpose equals i, same thing.

And then you look at this and say, wait a minute. We've got a 3 by 3 matrix equals a 3 by 3 matrix. That's 9 constraints. And that seems excessive because we only got 9 numbers. So if you take out 9 constraints, there's nothing left, no degrees of freedom.

Well, the thing is that this is a symmetric matrix. So it's actually only six constraints, right? Because the diagonal plus, say, the top right triangle is 3 plus 3 is 6. So there's 6 constraints. The 3 by 3 matrix has 9 numbers. We've got 6 constraints. 9 minus 6 is 3, 3 degrees of freedom.

So typically, that's where people stop. That's all the conditions we need on the matrix. But actually, it's not. You need an additional constraint, which is different in that it's not the quality of some number to some value. It's really a single bit of information. And why do we need that?

Well, here's the example of why we need that. So this matrix, if you take its transpose-- it's the same matrix-- and then you multiply it by itself, you get the identity. So this matrix satisfies that first condition, but it's not a rotation, right? Because it's flipping the z direction.

So it's a mirror image. It's I'm standing on the floor, and there's another person talking on the other side with his feet up. And that person's left hand would be my right hand and so on.

So no rotation would turn me into that other person. It's like turning your left hand into your right hand. It can't be done with just rotation. It can be done trivially with a reflection. And reflections satisfy this.

So in order to eliminate reflections, we have to introduce that constraint. So if, say, we set something up like a least squares problem-- we're trying to get these cameras lined up. And we make some measurements, and there's some errors that we're trying to minimize.

Well, then we also need to enforce the orthonormality constraint. And annoyingly, we also need to enforce this constraint. and that's hard. So that's one reason why we won't be using orthonormal matrices much.

OK. Oh, by the way, this means that the inverse of a rotation matrix is easy to obtain. I mean, we'll use some of these properties as we go along.

OK, so one way of thinking about this is in terms of a physical model where we have a cluster of points, a cloud of points that we've, say, got from the left coordinate system. Then we have a cluster of points from the right coordinate system. And we want to superimpose them. We want to line them up as best as possible.

And one model for that is, if this is one of the left-hand coordinate system points and this is one of the right-hand coordinate system points after we've transformed it, we want to make that distance as short as possible. And in terms of physical model, we can think of that as a spring that's connecting the two. Why is that?

Well, because by Hooke's law-- let's call this distance e for error. By Hooke's law, the force is proportional to the displacement from the rest length. Let's assume these springs have a rest length of 0.

And then the energy is the integral of that. So it's a half the displacement squared. And so we can think of this as an energy minimizing problem, which is what a physical system like that would do. It would want to adjust itself to minimize the energy. And we've run across that in a way in that last pattern-- not the last, the one before last where there was a translation and rotation in the scaling based on attraction of points to corresponding points and so on.

OK, so we can think of this and we can approach this in a sort of least squares way. So what we're trying to do might be this. So we define some kind of error.

So we choose to transform the vector in the left coordinate system by rotation and translation into the coordinate system of the right system, and then we compare it with what we get there. And ideally, they should be the same. And if they're not, then we get some error.

And now, this introduces some sort of apparent bias, that we're treating one of the two coordinate systems differently from the other. And that's undesirable. You know, like, why should one coordinate system have preference over the other? So we want to be sure when we're done to check that, if we had found the transformation from the right to the left system, it would be the exact inverse of what we found going from the left to the right system.

Otherwise, there's something wrong. And we already discussed this for example in line fitting. y equals mx plus c is not desirable because it pretends that there's no error in x and there's all the error and y when, in many of our applications, the error is in image position, which includes both x and y. So similarly, here, we'll be looking at transformations and methods that have that property that there's that symmetry. And that symmetry is not apparent in that formula, but so we'll have to actually check for it.

OK, so what we want to do is, of course, minimize that. And what do we have to play with? Well, we've got r, and we've got the translation r0. So that's sort of, in a nutshell, the problem we're trying to solve.

We've got a bunch of corresponding measurements in the two systems. And we're trying to find the coordinate transformation between the two systems using it. And again, as I mentioned, that's sort of the other way of using those equations.

Normally, you would be just going in the forward direction of the problem. You plug in your rl, rotate, translate, and you get rr where the rotation and translation are known. And what's not known is the correspondence between the coordinates of points.

And here, we're doing the very opposite. We're assuming that we know the coordinates of the points, pairs of them. And we're trying to find the transformation that makes that work.

OK, well, there are a bunch of methods we could use to do this. One of them is to separate the problem of finding the translation from the problem of the rotation. And actually, all of the methods we'll look at take that approach because it greatly simplifies the problem.

We, in each case now, only have to deal with 3 degrees of freedom at a time. And one way we can do that is to pick some sort of reference point in each of this point of clouds and go with that. And so here's a way of taking, say, all of the left points and constructing a coordinate system in it.

So let's suppose we have point 1 here. So this is a measurement of some point. And we're going to use that as the origin. So out of all the points we've measured, we pick one and declare that to be the origin-- OK, step one.

Then we look at a second point, and we connect these two by a line. And that's going to be one of our axes. Now, the separation between 1 and 2 won't be a unit. So we don't use the actual difference between 2 and 1 as our coordinate. We normalize it.

OK, so let's write it over here, x hat unit vector rl2 minus rl1. Oh, sorry. Let me just define x and then define x bar in terms of x-- save some writing.

OK, now, we take a third point. Now, we can't sort of connect these up and say, OK, that's the y-axis because then x and y-axis won't be perpendicular. But what we can do is decide that the plane defined by 1, 2, and 3 is the xy plane. So we can certainly do that.

And actually, then based on that idea, we can just remove, the vector from 1 to 3, the component that is in the direction of 1 to 2. And what's left is going to be automatically perpendicular to us. And that's the y-axis.

So we pick y equals rl3 minus rl1. And then we find the component of that rl3 minus rl1 in the direction of x hat. So we picked this vector from 1 to 3. But we take out a component that's in the x-axis direction, which we do simply by taking the dot product of that vector with the unit vector in x direction.

That gives us a scalar. And then we multiply that by the unit vector in the x direction. So that removes this component, the one that's parallel to the x-axis. And we're left with something that's perpendicular to the original.

And then, again, we can make a unit vector. So this won't be a unit vector in general, of course. OK. And it's going to be easy to show that x dot y dot is 0, that they're perpendicular to each other. You can check that very easily just by taking the dot product of this with our definition for x. Or you can just show that the y vector is perpendicular to the x vector, not the unit, really.

OK, so we have the origin. We have the x-axis. We have the xy plane. Yeah.

So we have an object, and we've identified quote "interesting" points on it. And we've measured them. So it's a cloud of points. And they could be corners, not edges. Because edges don't give us full 3D constraint.

So if our calibration object or whatever was a cube, we might very well have these points. Or if it's someone's head, then it might be the position of the iris of the eye and so on. And we've measured those in both coordinate systems. That's the important thing. But right now, we're just working on the left coordinate system.

OK, well, at this point, we define z to be the cross product of x and y because the cross product is perpendicular to both vectors in the cross product. So z is going to be perpendicular to both x and y. So now, we've got a whole coordinate system. We have a triad of unit vectors that define the coordinate system.

So we've done that for the left, and we only need three points. So if we measure the whole bunch of points, we don't need them. We just need three.

Now, we do the same thing for the right coordinate system. And you know, I'm not going to go through the process again. We'll just get an Xr hat, a Yr hat, and a Zr hat. And I suppose I should call those x left hat, y left hat, and z left hat.

So what I do is I take my two clouds of points measured in these two coordinate systems, and I use them to build an access, an x-axis, a y-axis, and a z-axis. And now, all I need to do-- quote "all I need to do"-- is to map those unit vectors into each other. So I need to find a transformation that puts xl into Xr and the transformation that puts yl into Yr and so on.

OK, now, because I've artificially removed a translation by picking one of the points as the origin, I don't have to worry about translation for the moment. I only need to deal with rotation. I've separated that problem. So I've got then Xr hat is Rxl hat, right?

So the unknown rotation R does this, and it also does that. I mean, if I just had that first line, that wouldn't pin down R because there are lots of ways of rotating one vector into another. It's not unique.

So that's what I expect. And now, my problem is solving for R. So I've got these three vector matrix equations, but I think you can see that we can compose them. We can stick them together into one equation.

So we could have Xr hat, a Yr hat, a Zr hat. That's now a matrix, right? Because in the representation we're using, these are column vectors. So this is a 3 by 3 matrix and the same for the right-hand side.

And I think you'll agree that this single equation is equivalent to these separate equations. Because if I multiply the matrix by that first vector, I get the first vector over here and so on. So that makes the answer trivial, right?

R is-- let's see, I need to post-multiply. Right. Because if I multiply both sides of that equation up here by the inverse of this matrix, it'll drop out over here. It's multiplied by its own inverse, so it gives identity.

And then over here on this side, I get this expression. Does the inverse exist? We always ask that question. When does this fail?

Well, it should. Because by construction, those things are orthogonal to each other. So you've got kind of an ideal case where the three columns of the matrix are orthogonal to each other as opposed to being linearly dependent. That's the other extreme end of the spectrum.

OK, so there we are. We solved the problem. So we said that, if we remove translation for the moment, we have 3 degrees of freedom left to find. And then we said, OK, we have three correspondences between the two coordinate systems. So that sounds about right, three constraints, three unknowns.

But those constraints are much more powerful because they are not just a scalar equals a scalar. They are vector equals a vector. So each of them is worth 3 points. So we actually have 9 constraints.

We're saying that point number 1 transforms into point number 1 in the other system. Point number 2 transforms into-- and so on. And so each of those are vector equalities, you know? Rl of 1 is going to map into Rr of 1 and so on.

So we have three vector equalities. Or if you like, how much constraint do you need to pin a point down in 3D? Well, three constraints, right? You need x, y, and z or some other three numbers, like distances from three-- I don't know-- Wi-Fi routers or something. You need three things.

So in this case, we've got 9. But wait a minute, rotation only has 3 degrees of freedom. So what's going on? It's excessive. So we don't need three points. We can just stop with two, right?

Well, but we said with two we just get the x-axis. And then the whole thing can still rotate about the x-axis, so that's not good enough. We do actually need all three.

But what's going on is that we are not using all the information we get, right? So let's make a tally here. So of the point number 1, we are using all three of its components. So this provides three constraints.

Then this one here, we're not using the distance from 1 to 2. We're only using the unit vector in that direction. So it's a direction. So it's really providing only two.

And then this one here is even worse. We're actually only using one constraint from that. And so the total is 3 plus 2 plus 1 is 6. Oh, well, that includes the translation.

So that explains some of what's going on, that this particular construction does give us a rotation, but we're not using the information from the points equally. So we've made some arbitrary decision. We picked number 1, and we're using all-- so that seems wrong, right?

Because I mean, who's to say which point's more important than another? I mean, there might be weight associated with it because, when you measured them, you found one point that's really distinctive and you're very certain about where it is. In some other matches, that might not be so good. But that's a different issue. OK, so that's one thing about this that's less than optimal.

So another one is what if I picked this point first? So this is my 1. And that's my two, and that's my 3. Will I get the same transformation?

No, I won't get the same transformation because I've selectively neglected information. And now, I am selectively neglecting in a different way. I'm throwing away something else. And so that also tells you that this isn't ideal.

Now, in practice also, we might be inclined to measure more than three correspondences. If we have these two LIDARs on the vehicle, they're giving us hundreds of points. And there will be some error in all of those measurements.

But again, as we know noted, if you have lots of points, least squares or some other method can dramatically improve your accuracy if you can make use of all of those correspondences. So this doesn't. It just uses three points. And yeah, you could now repeat it picking three other points and do that lots of different ways and averaging. And that would be better, but it would be kludgy, ad hoc.

OK, so ad hoc method number one, it's-- and in some sense, this might be a way to go if you needed an initial guess or if you needed a really rough answer. Say you had a more precise-- so this could be like pat quick and pat max where you use one to give you a fast approximate answer, and then you use that as initial condition for something else that gives you a more precise answer.

OK, so here's method number two. And we briefly talked about this in 2D, so let me just refresh your memory because we didn't really spend a lot of time on it. The idea was that we had some sort of blob, a binary image. And we're trying to make some measurement on it that could help us identify or align.

And we said that one way of doing that is to look at the axis of inertia, right? So if I have a cloud of points as opposed to some binary image, I can do the same thing. I can find the axes about which the-- oh, wrong way around. Excuse me-- minimum axis, maximum.

So why do I know that's the minimum? Well, because the inertia depends on the integral of the distance squared times the mass. Let me put r.

And so about this axis, these distances are all very short. So the sum of squares of those distances is going to be small, whereas about this axis at right angles, a lot of the distances are large. So I'm going to get a huge inertia. And you can show that they have to be at right angles to each other. And you can show that you can find them by finding the eigenvectors and eigenvalues of a 2 by 2 matrix.

OK-- so 2D. Let's go to 3D with the same idea. So there's some object in space, a bunch of points. So we've identified these points in a way that we can find them again in another image.

Now, we're trying to establish a coordinate system. Because we know that, once we've got a coordinate system ir the left data and the right data, we just have to correlate those two coordinate systems. And over there, we use that method to find a coordinate system.

But we could also look for axis of minimum inertia in 3D just as we do in 2D. And in 3D, we're then also going to find that there's a perpendicular axis that has a maximum inertia. And the difference about 3D is that there's a third axis, which is a saddle point.

So if you move the axis in one direction, the inertia gets bigger in a quadratic way. And if you move it in another direction at right angles, the inertia gets smaller in a quadratic way. So it's sort of like that, kind of hard to draw.

But here's the saddle point axis. And then if you move it in this direction, the inertia goes down, same on this side. But then if you move it at right angles, the inertia goes up quadratic. Anyway, so if we can figure out what those axes are, we should be-- well, let's do the math. It's not that hard.

OK, so what we're looking for is expression for the distance from the axis. Because, again, the inertia is the integral of that distance squared over the object. OK. So that means we need to figure out a formula for-- so this is the point r, vector r. And here's the axes. And we need to find that distance.

Now, I'm going to cheat a little bit because I'm going to pick the centroid as the origin, same trick. I want to separate the problem of finding the translation from the problem of finding the rotation. And we know in 2D that the answer is that it goes through the centroid. And then we had that formula with-- we knew the sine and the cosine of the angle, so we could compute the angle.

OK, so we're going to do the same thing here in 3D. Let me blow up that picture a little bit. So this is the distance r. This is our vector r. This is our axis. Let's call it omega hat is the direction of the axis.

And here is r prime. We drop it perpendicular on the axis. And what we want to know is the distance between r and r prime. So what is r prime?

So let's see, we need to find the component of r in the direction omega. OK, that's that component. And we're going to subtract that out.

No, I'm already one step ahead. Let me say r prime is just r projected onto omega. And this is actually our r minus r prime. So this is the length we're looking for.

OK, and so we need r squared. Don't worry too much about this algebra because we won't be using this again. But at least try and follow it, and tell me when I screw up.

So if I multiply this out, I get r dot r. And then I get minus 2 r dot omega hat squared plus r dot omega hat squared. That's because there's an omega hat dot omega hat. This is a unit vector, so that's just a 1. So I can just forget about that.

Well, and since the two things have the same form, so I end up with r dot r minus r dot omega hat squared. So that's my distance from the axis. So that's what's in the formula for inertia. And so I'm going to then say that the inertia-- and what I'm interested in is how that varies as I change omega.

So omega says, unit vector, they can point in any direction. And I'm looking for the directions where that value takes on an extreme one or maybe a saddle point. But starting off with just the minimum, I want to find where the minimum. And, well, that doesn't look particularly simple, but we can rewrite this.

So first of all, let me take this and rewrite it. Now, dot products are commutative. So I can write it this way. I can write that. And then we have this notation where we write the dot product in that form.

And these multiplications of the skinny matrices are associative, so I can write it that way if I want to. And this, of course, is the dyadic product, which we've run into a few times before. It's a 3 by 3 matrix, a very simple structure.

OK, now for my next step, I'd like to rewrite this also. So I'm going to say that's the same as r dot r into omega dot omega. Well, in fact, you know, I had this over here. I just dropped the omega dot omega because it's 1, but let me put it back in again. OK, then that is r transpose r omega transpose.

Oh, sorry. So we're going to do the double integral of that-- the triple integral of that. And so that I can write as r dot r into-- this I is the identity matrix. And I wrote the inertia with the blackboard bold I, so that it wouldn't get confused with this.

OK. So now, I've got the whole thing is omega transpose times omega. And I have two terms. I have this thing, which is the identity matrix-- the integral of r dot r times the identity matrix. And then I have this thing, which is rr transpose.

So given the points, I can-- well, I've written it as an integral. Of course, if we have a finite number of points, it would be just a sum. So this first part is just an identity matrix multiplied by some scalar. And this is a dyadic product. And so, what I'm looking for is an extremum-- of that whole expression.

Now, I've run out of different versions of i. So I don't know, let me call it A. So we have something of this form. And I'm looking for, let's say, a minimum. And even though it says A, this is called the inertia matrix.

So given some three-dimensional shape, I can compute this 3 by 3 inertia matrix. Or given a cloud of points in 3D, I can compute this matrix. This is just a detail of how to do it from the points.

And so what is that problem? Well, that's just the classic eigenvalue eigenvector problem, right? The axis I want for the minimization is the eigenvalue associated with the smallest eigenvalue, obviously. And then I can look for, also, the other axis that maximizes it. And that's the eigenvector corresponding to the largest eigenvalue.

And then there's a third one, which is in between. And those three axes are at right angles to each other. In some nasty cases, they're not, but we can make them in that case at right angles to each other.

So it's a very simple eigenvalue eigenvector problem. Now, finding eigenvalues and eigenvectors of 3 by 3 matrices is a little bit more work than 2 by 2, which we did by hand. But of course, you know, you've got all sorts of tools for doing this kind of stuff.

And there is actually a closed form solution. Why? Well, because polynomials of degree 3 do have a closed form solution even though we generally only remember the formula for quadratic.

OK, so what are we doing? So what we've done now is we've taken the point cloud in the left coordinate system, and we've built a coordinate system basis in it. We've got three axes that are at right angles to each other and just based on the distribution of points. And so we have a left-hand coordinate system based on these three vectors, the ones that minimize, the ones that maximize, and the ones at right angle, that saddle point.

Now, we do the same thing for the right-hand coordinate system, same construction, right? If there's an elongated point cloud, the x-axis is going to be along the length of that cloud and so on. And then we just use the method we had before. I guess it's gone, but the same method we used for method number one. And that gives us then another way of relating the two coordinate systems or relating the two positions of the object because those problems are duels to each other.

So this sounds a little bit more reasonable. Because first of all, it treats all the points equally. They are all thrown into the pot into these sums. We're not picking one and using all of its information and only using a little bit of the second point.

Also, we're using all the points. And we've constructed a least squares problem that performs some sort of minimization. So we're using all of the points. So what's the problem? Why is this not generally used?

Well, for certain kinds of problems, like lining up cryogenic electron microscope images, this is fine. This works. It's not always ideal.

And one reason is that it fails on the symmetry. Well, suppose I have a sphere. It doesn't have a minimum maximum and saddle point axis because its inertia about any axis is the same.

And then you might say, well, that's not fair. Sphere is completely symmetrical, so we have to accept that. So we might grudgingly say, OK, so it doesn't work for a sphere, but that's sort of reasonable.

Well, what if I told you that, if I give it a tetrahedron, it also won't work? It turns out, if you compute it's inertia matrix, it's inertia is the same in all directions and the same with the octahedron. So these are just special cases. I mean, there are obviously an infinite number of figures that have that problem.

Again, the inertia of this, surprisingly, is independent of the direction. Even more surprising, if I take a cube, now you'd think with a cube-- you know, there are three well-defined axes. So what's going on there?

Well, the trouble is that the inertia about those three well-defined axes are all the same if it's a cube. If it's a brick, then you're fine because the three axes will have different inertias. And you can line things up that way.

But with the cube, again somewhat surprisingly, you can pick any axis, you get the same inertia. OK, so what's going on? What have we lost? What have we forgotten?

Well, we haven't used all the correspondences. Well, in this particular ad hoc method, we have taken a point cloud. And we've sort of found its elongation and we've built these other axes without thinking about the other point cloud at all. Then we've done the same thing for the other point cloud.

But we actually-- assuming that we know the correspondences. So we know that this point is here and that point is there. And that's a very different problem, right? Because even with a sphere, I can line these up correctly if I have the correspondences.

So this is kind of a two-edged sword because we'll see that there are some methods that don't depend on correspondences. They're very handy. They're very appealing. Because if you screw up the correspondences, they're not going to be affected.

On the other hand, you often run into this type of problem where the method won't give you an answer because of some symmetry. And so they're generally not as accurate as methods that do take into account-- OK.

So enough of these ad hoc methods. I bring them up partly because people actually use them and often don't know that there's a problem with them. That's one reason. And the other reason is that sometimes these are useful to give a quick answer, and it's systematic. I mean, there's no iteration or search or anything. You just compute the eigenvalues and eigenvectors, which, for a 3 by 3, is basically a closed form operation.

OK, so let's go back. So the problem is not the translation. The translation is going to be pretty easy to deal with. So let's focus on the rotation.

So generally speaking, we have a bunch of different methods of representing rotations. So let's, first of all, look at the properties. OK, preserve dot products, that means-- right? That's sort of obvious.

And that has two sort of corollaries. One of them is that preserve length, which is just what you get if you make a the same as b. And they preserve angles, right?

Because, for example-- and what's interesting is they also preserve triple products, so R of a.

Now, the triple product of the three vectors is just the volume of the parallelepiped that you draw based on suppose this is vector a. This is vector b. This is vector c. You can imagine drawing this three-dimensional shape, which has parallel surfaces. But it's generally not a brick. It's skewed generally. And this is the volume of that object.

And now, imagine taking those three vectors and rotating them all with the same rotation. Well, that just rotates this figure. And it'll retain its volume, right? And that's this quantity. So that makes sense.

What's important is that it doesn't flip sign. If we had replaced that rotation by a reflection, that triple product would have flipped sign. And you would say, well, what's a negative volume? Well, it just means that you don't have the vectors in the sequence of the right-hand rule. You have a left-hand rule.

So you know, I was talking about reflection in the ground reversing z. Well, obviously now, if I take that triple product-- I suppose I should write it out, a dot b cross c. And if I flip the sign of one of these, I'm going to get a negative sign.

So when I say volume, we should really take into account the sign so that the magnitude of that triple product is the actual volume. And the sign of it tells you whether it's been flipped from left-hand to right-hand coordinate system. While I have this here, let me just make sure we understand that there are a large number of equivalent ways of writing the triple product.

So in terms of our earlier discussion, all of this corresponds to orthonormality. And this corresponds to that condition about the determinant being greater than 1. OK, so those are all the properties we need to proceed.

So now, we set it up as a least squares problem. We've got correspondences between vectors measured in the two coordinate systems. Then we're trying to find the transformation between them. to find some sort of error.

OK, do I want to start with the [INAUDIBLE]? Oh, OK. So we get rr. So that's the transformation we're dealing with. And we can then define an error for the i-th point.

And of course, what we're trying to do is minimize that error. And what we have to choose is the offset and the rotation. OK, so again, we're arbitrarily picking the left coordinate system and mapping it into the right coordinate system and then comparing the result. And they should be the same, ideally. There will be some small error in practice.

So we're trying to make that error as small as possible. And we try and find the parameters of the transformation by minimizing that error. OK, now since I know the answer, I can transform things in a way that simplifies the problem.

So I just take all of the coordinates in the left-hand coordinate system, add them up, divide by n to get the centroid. And I do the same in the right-hand coordinate system. And I pick those as the origins, basically. So I'm going to subtract out that-- OK.

And this is in line with the idea that I want to separate the problem of finding the translation from finding the rotation. So this way I'm going to get rid of the translation. We'll worry about it later. It's very simple.

OK, by the way, while we're here, we might wonder what that is. So I've changed all the coordinates to be relative to the centroid. Now, I take them all and add them up. What do I get?

Well, you could plug this in here, and you're going to get n copies of that. And you get the sum of these. And then you look at that formula. And when you divide by n, they all cancel out, right? And it's a vector.

So one feature of moving through the centroid is that now the centroid's at the origin of in new coordinates. That makes sense, right-- and similarly for the other one. And we need this property in the second cell. OK.

So I'm going to plug these new coordinates into my error formula. So to do that, I need to solve this equation for r sub ri. Well, it's hardly worth writing out. It's just moving this to the other side.

And if I make those substitutions, then I get this where the offset now is different. And so my new problem is-- so there's a square. And I can multiply these out, but I'm going to group them.

So I'm going to treat this as the difference of that thing and that thing and then undo the square. And I get-- now, there are other ways to get here. You can differentiate with respect to the translation, but I think this is the most obvious.

OK. Now, over here, this whole center term goes away because it involves the sum of r of sub ri. That's 0, right? And then it involves the sum of the rotated version of r sub li. And we know that sum is going to be 0 as well. So this whole term disappears. So that makes life a lot simpler.

OK, so then the next question is, what should we choose for the offset for the translation r0. Well, this term doesn't depend on the translation at all. So forget about that one for the moment.

This one does. It depends on the square of it. So how do we make it as small as possible? 0, right? So now, you can see a couple of things.

One of them is over here we find that the optimum solution has r0 prime 0, which means that this is the case. r0 is-- so this is the formula for the translation. Now, of course, right now, we can't use it because we don't know what the rotation is. So we don't know what big R is.

But it's very intuitive. It says that the translation is the difference between where the centroid is in the right coordinate system and where the left coordinate system centroid is after you rotate it. So the centroid maps into the centroid.

So we've accomplished one objective, which is to separate the problem of finding translation from the problem of finding rotation. And then we have a formula there. If we ever find the rotation, we just go back to that formula, plug it in to find the translation.

OK. And then what's left is-- so basically, that gets rid of this term as well. So that's gone. So we're left with this part and only worry about rotation.

So our job is to find-- so our new error-- right? That's just that error term. And we're now going to minimize.

And that means-- and I'm just taking this term and squaring it. I suppose I can write this out here. This is dot. So I'm just expanding this out.

And so from the first two terms here, dot product of these two gives me that. And then I have the dot product of that with this. That gives me this part, which occurs twice. And then, finally, I have the dot product of these last things.

And since we said that rotation preserves length, I might as well compute the length before rotation. So that's very convenient. So these last two collapse into that, and r doesn't appear in there.

So this is fixed in the sense that it doesn't depend on the rotation. This is fixed in that it doesn't depend on the rotation. And so we now focus on this term except it's got a minus sign.

So instead of minimizing, we want to maximize this. So that's sort of, in a nutshell, our remaining problem. And if we can do that, then we're done.

And notice how this intuitively makes sense. Think of the cloud of points, and there's a centroid. Now, connect each of the points in the cloud of points to the centroid.

So there's a vector from the centroid out to each point, sort of like a spiky-- what's the term in sushi? Uni, that's it. It's uni. It's a sea urchin, right? It's got these spikes, OK?

And so what we're doing here is we're taking one of these sea urchins and we're bringing the other sea urchin into alignment. And the way we're doing it is we're saying that corresponding spines should have a small angle between them. Their dot product should be large, right? Because the dot product between two vectors is proportional to the cosine of the angle. So the largest you can make that is 1, which happens when theta equals 0.

So this makes perfect sense because we're taking each of the corresponding spines of this spiky ball and saying, put a contribution in that's proportional to the dot product and rotate it to make that as large as possible. So our whole rotation problem has been simplified down to this if we can solve that. And so, well, we are calculus people. So one thing you might think of is-- we've done this kind of thing before-- we'll differentiate with respect to r and set the result equal to 0.

And, well, that's not quite right because r is not a scalar. It's not even a vector. We know how to differentiate with respect to a vector. It's a whole matrix.

But we could just take the 9 elements of the matrix and string them into a 9 vector and differentiate with respect to the 9 vector. So we could do that. The problem is those 9 numbers aren't independent. They have to satisfy all of those annoying constraints. And so it's actually very hard to impose these constraints subject to R transpose R is I and determinant of R is plus 1.

So you can go down that route, but it gets incredibly messy. And so we won't be doing that. We'll be looking at other representations for rotation where this problem comes out very easily.

But it's like a lot of things where there's a bit of an effort to develop the technology, the representation. Once you've got the representation, it's trivial. So we'll spend the next class building up that representation. OK.