[SQUEAKING]

[RUSTLING]

[CLICKING]

BERTHOLD HORN: "Quick," meaning pattern recognition quickly. And that's in distinction from another pattern we'll look at later, which is slower but gets a more accurate answer. So a number of terms were defined there. One of them was that of a model.

So there's a training step that produces a model. And the model consists of probes. And the probes are places where we will be collecting evidence for a match to produce a scoring function. And we think of that score as a surface in a multidimensional space which has as its axes the various degrees of freedom. And we're looking for a peak in that score surface that's above a threshold as a way of indicating that there's an instance of that pattern in the image. And there may be more than one, and there may be none.

And at the highest level, we look at quote, "all poses," meaning we're going to quantize them, but we have to consider the full range of rotations, translations, scaling, whatever. And so the complexity goes exponentially with the number of degrees of freedom. So even if we quantize fairly coarsely, say somewhere between 10 and 100 steps, that means we're multiplying the amount of work by 10 and 100, something like that.

So normalized correlation was expensive and couldn't handle more than two degrees of freedom, translation. Well, this method is looking at very few points in the image. And so it can afford to deal with a large number of degrees of freedom.

OK. That's a great question. So at one level, we're building up. So we're starting off looking at very low-level issues, and then we're building on top of them. Then as to why did we pick these patterns, well, one reason is that it's a package of things that are related. So we don't have to re-learn terminology and whatever.

So for example, the front end for this one, as you saw, is the pattern we talked about before. And the next one we're going to talk about is closely related to this. It uses this one to get a first guess at the answer. And then the one after that is a fast way of computing a step that we need to do the subsampling.

So is it a random sampling? Well, kind of, because machine vision has exponentially grown. And there was a time where there wasn't enough material to fill a whole term. So I taught this course, which I called "Making Machines See and Feel," because half of it was vision, and half of it was robotics manipulation. And we can't do that anymore. Now we've got multiple courses on machine vision.

So this is a particular approach to machine vision, which some people refer to as the physics-based approach to machine vision. By the way, they made me change-- the head of department contacted me and said I couldn't have a frivolous title like "Making Machines See and Feel." And that's why now it's called "Machine Vision."

So the patterns we're looking at hang together, which makes it easier to discuss them. Their patterns of the most successful machine vision company. So there's some weight there. Is it what they're doing today? Well, a lot of the machines they sell do this. Others do other things, which we have yet to find out, because we don't know their trade secrets.

And the things we're talking about in the pattern discussion is mostly 2D. So it's restricted to situations where we're dealing with two-dimensional surfaces, like integrated circuits, printed circuit boards, conveyor belts, the ground outside an autonomous car, things of that nature.

So after this, we'll go on to higher level still. So it's part of a progression. And yes, we're not covering everything possible. And so we're taking stabs at things that give you a feeling for what's there rather than try to cover everything.

So in this-- back to this. So there's a training step where we show it an image, and it automatically computes this model. And that was very important for them, because we can send someone to MIT and get a degree and handcraft some code to do something. But that's only worthwhile if you have a huge application for that.

I forget what the example was-- pencils or something, earphones-- toothbrush, sorry. There was someone who was asked about supporting startups, and his test was the toothbrush. So if he can sell a billion of them, then maybe he'll fund it. But if for every visual task you have to expend that effort to program it, that's not good.

So the training method is very good, because someone with not a great deal of experience can show the system the object, hopefully a good sample of it, and get a training image. There are some refinements you can do. As we discussed, for example, you can have negative weights. But those are things that a person has to-- can't think of an automated way of doing that yet. And so that's actually something that's never done, because it's much easier to just go with that if it's satisfactory.

So what do we do with this model? Well, then we map the model onto the real-time image. And in the process, we use the pose, which is translation, rotation, and all these other things. So our model now is mapped on top of the image.

And then at each of the probe positions, we look at the brightness gradient. So actually, we pre-process-- we don't really go back to the image. We go to the brightness gradient. And we perform a test to collect evidence, to collect evidence, that this object is actually there. And the evidence is cumulative. So we just add it up.

And so that's the idea of doing a lot of local operations, where the final result is just the sum of those, a little bit like the binary imaging processing-- binary image processing methods that I mentioned. OK. And then we do this for quote, "all poses," meaning that we need to quantize the pose space. And that's where the limitation of this method comes from, which is that we will be limited in accuracy by the quantization of pose space that we have.

So what are all of these components of pose? Well, obviously translation. And when we were looking at the pattern, there was a whole page. But they were redundant in that we had a scaling one that worked in a logarithmic scale, and one that worked in the linear scale, and so on.

But let's just list the ones that aren't redundant. So there's going to be translation, rotation. And again, depending on the application, not all of these may be necessary. Then there's scaling. Now, scaling may not be necessary, because maybe your objects are all the same size, and maybe you're not dealing with the change in size due to change in distance, because you're using a telecentric lens so there won't be a change in size. But we allow for that.

Then there's skew, where, if you like, instead of rotating both x- and y-axes by the same angle, we rotate one by a different angle. And again, that may or may not apply in a particular situation. And then we have aspect ratio. So in a way, this allows a certain degree of generalization.

So you have a pattern. But if you want to, you can also deal with the same pattern in a different size, the same pattern that has been squashed in one direction and maybe extended in the other direction. So translation, two degrees of freedom, rotation one in 2D, scaling one, skew one, aspect ratio one, so a total of six independent parameters.

And so we're dealing with a potential six-dimensional space. And if you quantize that even to 10, then you are up at a million. And if you quantize it to something more reasonable like 100, then you're up to 10 to the 12-- really? Yeah, 10 to the 12, which is impractical. So in most applications, we don't use all of these.

By the way, if we put them all together, we get a general linear transformation. I'm doing this partly because when we get to 3D, it'll be handy to be familiar with these terms. So that's a general linear transformation. We've got six parameters and affine. It's an affine transformation.

And we could say, you know, forget this categorization. We're just going to think about the transformation in term of those six coefficients, which is fine, except that for us people, it's easier to think about rotational, translation, and scaling, because the individual-- what does it mean if I change a1,1, and I keep the other coefficients constant? And we'll talk some more about that transformation.

OK. So I want to talk next about the scoring functions. So it's clear what we're doing. We create this model. We map the model onto a runtime image. We collect evidence. We get a score.

We find the peak in the score surface in that multidimensional space. And if it's above a threshold, then there's a potential candidate-- there could be more than one, or there could be none. And so that's the overall process at the top level, where we explore the whole space.

And we talked about the way we perform the scoring, which is mostly based on the direction of the gradient. So. But it can take into account the magnitude of the gradient, as well. And we had this issue that it's possible that we get a match even when the we're not on the right part of the object, just because there's a certain range of angles which are going to be accepted. And so there's going to be some probability that a random gradient in the background texture is going to match, and we--

So this was the function used for grading the quality of match in terms of the direction of the gradient. And this was, of course, just a sample. But it's the one that's highlighted in the pattern. And for this one, this is 3 over 32. So there's roughly a 10% chance that even if we plunk the probe down in some totally wrong place that we'll get a match.

And so in the pattern, this is referred to as noise. A little-- I guess it's background noise. I don't know, it's a little confusing. It'd be better if they didn't use that term.

If we have the version that ignores polarity, then, of course, it'll be twice that. So that's one disadvantage of that version of the matching function. It allows you to reverse the contrast. But at the same time, the noise contribution goes up.

OK. So then they define a number of scoring function. And they're kind of trade-offs between accuracy, speed, whether it's normalized, whether the actual value of the result is meaningful, or whether it's just something that's bigger if you have a better match. And so let's look at that. So we've got--

Now, if you remember, the probe was something that had a position, a direction, and a weight, and possibly some other stuff. But those were the top things. So here is the weight of the probe. So we stepped through the probes. I stepped through the probes, and that's the weight.

And what does that do? Well, it discards things that have negative weight. So just in case you've used that capability, for this scoring function, you throw those out. This is R dir, the direction scoring function. And it's looking at the difference between the angle, the direction of the probe-- so that's the direction of the probe-- and the gradient at the position that the probe is placed in, position--

And big D is just a function that gives you the arctangent of ey over ex. It's the direction of the gradient in the runtime image. So let's look at that. So what happened to the other components of the pose?

Well, at this point we're dealing with compiled probes. So here we're just varying the translation. We're only varying the translation a. We've already mapped the probes according to the other components of the pose. Yeah?

Oh, that's a vertical bar, meaning absolute value. Sorry. And in some sense, that's just to take account of the fact that this wraps around. And we could have done that some other way, like mod 360 or something.

OK. And this version is normalized in that we divide through by the sum of weights, such that if we get a perfect match the result will be 1. And so the absolute value of this score, of this particular score, has a meaning. It's not just that it gets bigger if you have a better score.

And this is used in the coarse step of the algorithm. What else can we say about it? Then there's a second version. What's the other version? It's 1b. Sorry, this is S1a. This is S1b.

And this one doesn't multiply by the weight. And this is a slightly weird notation, but it's kind of clear once you think about it. So this is just a predicate that computes whether the weight is greater than 0, whether it's positive weight. And it's 1 if that's the case and 0 if it's not the case.

So that's just a funny way of saying we're only going to process probes that have a positive weight. And the advantage of that is we don't need that multiplication. And so it's cheaper than the other one. And so this is faster.

We haven't done anything with that quote, "noise term." So when we get to the preferred embodiment--

So it's very similar to 1b, except that now we're saying, well, if we apply this in a random place in the image, we'll get a non-zero result because of this random matching. And so by subtracting out that component N, which is an estimate of the number that will be matched randomly, we can improve the result. Yeah?

AUDIENCE: What's the D function?

BERTHOLD HORN: This function, OK. So D is the function that tells you what the gradient direction is in the runtime image. Gradient-- sorry, gradient direction at a plus p i. So I didn't underline these, but they are vectors. a is a translation, and p i is a vector that's the position of that probe, so.

And so then when I subtract the little di, get the error in the direction. And then I use that to access this function to decide how much to penalize the result based on that error.

OK. So this one now has the feature that if you're throwing the probe-- throwing the model down at a random place, on a random texture, the answer will tend to be 0, because we've taken out the random coincidental matches. And so unlike this one, where, yes, if you have a perfect match you get a 1, but if it's just garbage background, you don't get a 0 you get whatever N is.

So this is slightly better in that respect. It's slightly more computation not much. So that's the quote, "preferred" method. And if your implementation doesn't use that, that doesn't help you because they don't say that you have to use that. They give you other alternatives. And they just say that, well, this is the one that we think is the best.

So OK. So then we get as S2. Now, this one is not normalized. So it will not-- its absolute value will not be significant. It's just going to be larger if you have a better match.

And so what's this? So this is similar to this function, M of a plus p i is the gradient magnitude, which we haven't used so far. So this one actually takes into account the gradient magnitude, and it uses the magnitude directly. And it means there's another multiplication. And this version is used in the-- first of all, it's not normalized and is used in the fine scan step.

And then finally-- so a lot of this is used in getting to a potential candidate solution. And then there's a fine scanning step-- there's a scoring step that gives us an actual value. And this one is more work to compute. So this one, again, is normalized.

And it uses the magnitude of the gradient, but it uses it according to the scoring function that we defined, which was that it saturates. So that's the direction scoring function, and this is the magnitude scoring function. So in S2 a, we would just keep on going up, multiplying by the magnitude. In that one, we limit the contribution. So you don't have one single very good edge dominating everything else.

OK. And there are lots of details on how to make this run fast and so on, which we won't go into. I do want to talk about a couple of interesting points. Another thing we're not going to talk about is the granularity.

So we mentioned this, that we might work at a scale where these computations are cheaper as long as we can get a satisfactory result. And mind you, what's a satisfactory result here is very different from some machine vision, where if you can improve the recognition accuracy from 71% to 73%, you've got a paper. This is more like this has to work 99.9% of the time.

And so granularity is determined basically by decreasing the resolution until it doesn't work well enough anymore. So if for your task you know what "well enough" is, you can perform that task. And basically, for each granularity you have to go through the whole process. You have to build the model, because the probes will be different depending on the resolution. And then you run through, and you see how well it works. Yeah?

AUDIENCE: When wouldn't you want to normalize?

BERTHOLD HORN: Mostly when you're in a hurry. So it's just a computational issue. If you're at a part of the computation where you have to do it for every possible pose, you might want to save that step. If you at the end, where you near the correct answer and you only have to do it a few times, then you can normalize.

OK. So the granularity-- and they have a very sophisticated method for doing this. And I think we've seen enough of that part of the pattern, so I'm not going to go through it. But it is, obviously, in practical terms an important component, because you want this to run as fast as possible.

If you're at Amazon, and you watch the boxes come down the conveyor belt, and it's reading all kinds of stuff off the box and determining its position and orientation, you realize that you don't have a lot of time to do the computation. These things run at a hundred frames a second. And all of this computation can be done in that time in a relatively cheap processor.

OK. But I do want to talk about a couple of minor issues that may not be obvious. One of them is getting the directions right. So we're really focused on gradient directions. So we need to make sure that when we perform these transformations that we transform the gradient direction correctly.

And so here's the issue. So here's my edge. This line is the iso fold. And here's my gradient, which, of course, is perpendicular to the iso fold. And now suppose that I have an operation that changes the aspect ratio. So I suppose I squash it. So I'm changing the aspect ratio.

Then of course, this line is going to have a lower slope, because the top is moving down, the bottom is moving up. So OK. And hmm, the gradient also is going to have a lower slope. Right, so it's no longer perpendicular to the iso fold.

So that is kind of obvious. And now if you're transforming x and y, it may be that the derivatives with respect to x and y transform in a different way. And so what do we do about this?

Of course, this doesn't affect translation. This doesn't happen for translation, doesn't happen for rotation. You rotate, and the right angle is preserved. Doesn't happen for scaling. But it does happen for the other two.

So how do we deal with this? Well, we have the gradient direction. That's our input. That's what the box we have in front of all of this computes. OK. So that's where we start.

And now all we need to do is compute the iso fold. So from the gradient direction, we get the iso fold. Then we transform that. And then we construct something at right angles to the iso fold. So here's our new gradient direction.

So the solutions are kind of obvious, but it's something that one might easily forget. How expensive is it? Well, not terribly, because 90-degree rotations are simple. So cosine of 90 is 0, sine of 90 is 1, minus sign.

So they just-- we don't even need to do multiplication. We just interchange x and y and flip the sign of one of them. And then after the transformation, we have to do the inverse, which is the transpose of that matrix. So again, we just interchange x and y and flip the sign of one of them and we're done. But it's something that's easy to forget. And until you run into the last two operations, it doesn't matter.

OK. Another thing which was almost like an add-on that they thought of, because it's not discussed much in the specification but it shows up in the claims-- and there's some discussion at the end of this, which is inspection. So first of all, the main focus here is figuring out where something is-- position, pose. A second aspect is recognition.

And how does that work? Well, you just look at the score. If it's close to 1, you've got it and again, this may occur in more than one place in the image.

If you-- typically, "recognition" means that you're distinguishing different things like cats and dogs. And so how does that work? Well, you have a library of models. Suppose that you have different types of screw heads, then there's a model for each of them. You run this process, and you see where there is a match. And if necessary, you can compare the match score to figure out what type exactly it is.

So that's recognition and position. And they also talk about inspections. So inspection is based here on a fractional match-- runtime image.

OK. So the model consists of probes. We put the probes down on the image. And for each place, we can determine whether it's a reasonable match or not. And in the process, we can calculate a percentage. And if the object is partly obscured, well, then that fraction will be lower. If, say, you're looking at a gear and some of its teeth are missing, then you'll get a match, but the score won't be 1. And so that's one direction.

The other one is-- so we can look at the runtime image, and see where it has edges, and then discover how many of those are actually matching something in the model. Now of course, if there's clutter and there's some background, then there would be a lot of edges that will not match the model. But that's also useful information. So that's a way of measuring clutter, basically that. Yes. And there are some more things to say about that, but that's the basic idea.

OK. A couple of more things I want to do here, partly in preparation for the 3D work we'll be doing. And that's to elaborate a little bit more on these transformations and to vaccinate you against some bad ideas, hopefully. So let's go 3D but in a kind of simplified world.

So we know already a lot about that. We know about perspective projection. So let's think about the projection of a plane. And this comes up a lot. I mean, we could be talking about tabletop, conveyor belt, integrated circuit, printed circuit board. And it's all in 3D.

Now, we can carefully line up the optics and so on so that we get an orthographic projection, in effect. Or we can deal with the real full 3D world. Could also be the road in front of a car. Lots of examples of flat surfaces.

OK, so two things. One thing we remember is good old perspective projection. And the other thing is that there is a camera coordinate system where this applies. And then there's other coordinate systems. Let's call them world coordinate system.

And what's the relationship? Well, there's going to be a translation. The camera has its coordinate system has its origin at the center of projection. And the world has wherever. If it's a robot arm, it might be the base of the robot arm. If it's a box, it might be a corner of a box.

And then there's a rotation, so in world coordinates and in camera coordinates. And we talked about how that simple formula for perspective projection only applies if we're dealing with a camera-centric coordinate system. OK.

And this matrix here is an orthonormal matrix that encodes the rotation. And we'll talk quite a bit about that later. OK. Now, what I want is the transformation from world coordinates, object coordinates, to image coordinates. And so I combine these two.

So what I'm going to get-- So this is just multiplying out that matrix. And this is, again, the same thing. So as we know, perspective projection has this nasty property that it involves a division that's nonlinear, and it's kind of a bit messy.

So that's a general case. And we'll find ways of dealing with that. But I want to look at the particular case where the thing we're looking at is planar. And that means that we can erect our coordinate system in that object. Here's our planar surface.

Rather than pick some arbitrary coordinate system, let's pick one where z is 0 so we only have two of these coordinates to deal with. So what we're sort of doing is we're kind of halfway between 2D and 3D, because we started off looking at a 2D surface embedded in 3D. And where we're going is some mapping from the 2D surface out there into the 2D surface and the image. Yes?

AUDIENCE: Where does the e w come from, the top-- numerator?

BERTHOLD HORN: E-- sorry, which-- e w?

AUDIENCE: Oh, yeah, next to the yw is a e w. Or is that a z?

BERTHOLD HORN: Oh, sorry. Oh, here?

AUDIENCE: Up top.

BERTHOLD HORN: Oh, sorry. That's my bad writing, yeah. zw. OK, and that's the one that's actually 0.

So that, when I do that matrix multiplication then, that means that I can ignore the third column, because it's going to get multiplied by zw. And so I can make that anything I want. So this column here doesn't matter, because it's going to get multiplied by zw.

And then I can kind of conveniently fold in the translation. So one of the annoying things about Euclidean transformations, real-world movement, is there's rotation and there's translation. And it's hard to treat them as one thing. It'd be nice to have notation, some magic thing. Let's call it a glob.

So the glob, you multiply it by your vector, and out comes another vector. And that single thing encodes both translation and rotation. So the way we've written it there, we've split those two.

Well, it'd be really nice to have them as a single operation-- multiplication of something, for example. Well, we can do that here because if we drop out that third column, we can bring in the x0, y0, and z0, and we end up with-- let's see. So we've got-- I called it r1,1. And--

So you see what we've done here. We've taken advantage of the fact that we don't need that third column for zw, because zw will be 0. And then we can fold that addition in by just multiplying that last column by 1.

So in this particular case, where we're dealing with a planar surface out in 3D, we can fold rotation and translation into a single matrix. And let-- call this matrix T. Let me call that matrix up there R.

So R has some special properties that we'll talk about later. One of them is that it's orthonormal. And that means if you take its transpose and multiply it by R, you get the identity matrix. And the other one is that its determinant is plus 1.

T doesn't have those properties. So that's very important, because how many degrees of freedom do we have? Well, in 2D it was a little easier to figure out. We had two for translation, one for rotation. So those already give you three degrees of freedom. And it's pretty obvious how you could represent them-- some x offset, some y offset, and some angle.

In 3D, it's a little bit harder. We have three degrees of freedom for translation-- a change x, a change in y, a change in z. And then there's rotation and the various ways of thinking about that, which we'll get to later. But basically, there are three rotations. There's one that preserves the xy-axis, there's one that preserves the yz-axis, and there's one that preserves the zx-axis.

And as a mental shortcut, you can say there's rotation about the x-axis, rotation about the y-axis, rotation about the z-axis. That's actually a very bad way of thinking about it. But it gives you the right number, which is 3, so.

OK. So that means that if we have translation and rotation, there should be six things-- six degrees of freedom. And then you look at that matrix up there and stuff, and there's already nine just in the matrix, and then there's another three for the translation. So there's 12.

So there's a redundancy. There's something wrong. We've got many more variables than there are degrees of freedom. And so that's because there are these constraints. So that R matrix isn't just any old 3-by-3 matrix. It has to satisfy all of those constraints.

And it's the usual thing. You have a number of variables, a bunch of constraints. You subtract them, and you get the degrees of freedom. So in this case, when we subtract out the constraints, it turns out we get 9 minus 6 equals 3 for rotation. The 9 is because it's a 3-by-3 matrix, there are nine numbers.

What's the 6? Well, the orthonormality constraint provides six constraints. Why? Well, each row has to be unit size vector, so that's 3. The rows have to mutually be perpendicular. Well, there are three ways of pairing them up, so then 6. So 9 minus 6 is 3.

OK. So then we get to this. And you know, this is an interesting way of dealing with perspective projection in the case of a planar surface. We just have to be a little bit careful. So suppose, for example, that we determine this matrix T experimentally by matching up an image of a calibration object like, I don't know, cube, or checkerboard, or whatever. Somehow we find this 3-by-3 matrix, and then we're done.

Well, no. Because this is a matrix with nine independent elements, which is way more than the transformation deserves. And so actually, we need to enforce some constraints. For example, it's derived from a rotation matrix. So that first column should be a unit vector. The second column should be a unit vector.

The third column we lost. We don't know what that is, although, of course, we can reconstruct it. Because it's perpendicular to the first two columns, so it's their cross product. That's, by the way, a useful programming trick. You don't really need to keep a 3-by-3 matrix for rotation. You just keep two of its rows or two of its columns, because you can always compute the other one. And in fact, in a sense, it might be more consistent to work that way.

OK, two constraints. Well, there's another one. These have to be orthogonal. So r1,1, r1,2 plus r21, r22, plus r31, r32, has to be 0. OK. So this matrix T should really satisfy those three constraints. And let's add things up.

So it has 3 by 3. It has 9 elements. Three constraints, that leaves us six, six degrees of freedom. That's correct. So that's all true and wonderful. It's just that typically, we don't enforce that constraint.

Why? Well, because the rest of it you can do linear least squares to fit to your calibration data. But how do you enforce these second-order equations? And so you will see a large number of publications that expound this approach, which would be valid if only they had enforced this.

So that's like a kind of attempt at generalization from what's covered in this pattern, namely 2D transformations, to where we'll be going later, which will be 3D transformations, and a kind of warning about this stuff that we have to be careful. So by the way, if I'm going to use this to predict where I'm in the image, I'm just going to use-- of course, the equations, I'm just repeating the equations from up there.

And so that means that if I take the camera coordinates and multiply them by some arbitrary non-zero factor, nothing changes. And of course, that's the scale factor ambiguity we already talked about. What does that mean? Well, one thing it means is that you can take this matrix and multiply it by some arbitrary constant, non-zero constant, and nothing changes. So that's another clue that this is a funny kind of matrix.

And yes, you could try to adjust its size by trying to impose these constraints. But-- OK, oh, this is called-- I should say this is called homography. And we'll talk later about its use in photogrammetry or misuse in photogrammetry.

OK. The other thing I wanted to talk about was, you may remember right when we started talking about this pattern, I said, well, there's prior art. And we listed blob analysis or binary image processing, and we listed binary templates, and we listed-- what else? Anyone remember what else we listed?

And there was a fourth one, which we didn't discuss. The other three we did discuss. And the fourth one is Hough transform and its generalization. So let's just briefly talk about that.

So that's because we picked our world coordinate system so that in the plane-- it's only working for this plane. And in that plane, zw is 0. What it means is that, suppose I want to deal with an image of this table, and then I want to refer to where my phone is on the table. I'm going to pick a coordinate system that's, say, lined up with the edges.

But the important thing is that z is perpendicular to that surface, so that whatever I'm-- where is this paperclip? I only need to give x and y, because if it's in that plane, z is 0. And so this will not apply if I have a true three-dimensional object. It's only the case where I'm confining attention to a plane.

And so it comes up in industrial machine vision, because-- in many ways. One is, obviously you want to try and get your optical axis as perpendicular as possible to the surface you're interested in, whether it's a conveyor belt, or the surface of the road, or whatever. But inevitably there will be some small error, and inevitably someone's going to bump into it and change it slightly.

And this takes care of it, because this slight generalization allows you to deal with other camera orientation so that you don't have to have a perfect orthographic projection. You can have a slightly skew one.

Thinking of other applications, in some countries there's like a surveillance camera at every intersection, and it's looking down at the road. Well, if you're mapping points on the road, you don't need the full 3D transformation. You just need this.

But you just have to remember that now your world coordinate system has the constraint that it has to be lined up with the surface of the road so that zw is 0-- assuming the road is really flat, but. OK.

Hough transform. So when I was a student, which is centuries ago, NASA had a large part of one of the Tech Square buildings, one of the original Tech Square-- they've since been renovated, so. I guess it was 535.

And there was a floor or maybe two floors dedicated to the following thing. At that time, people were very interested in various kinds of processes of elementary particles. And you studied them by shooting them into a cloud chamber, Wilson's design of a cloud chamber.

You shoot them in, you quickly decrease the pressure, and it's saturated with some vapor-- alcohol or something. And the ionized points in that space then form nucleation points for the liquid. And then you take a picture of it. And you find that there's particles zooming through in a certain direction. And then you can see how that depends on the magnetic field that's applied. If it's charged particle, it'll be curling around.

And so people spent many, many, many man-years and women-years poring over a table like this, where these images were projected down, and then they were fitting lines to them or arcs. And so there was a huge interest in automating this process, because it was clear that it was only going to get worse. Because the higher energy you get to, the more pictures you have to look at before you find the interesting stuff, and the more complicated the pictures get. So it was-- doing it manually was hopeless.

So edge detection, line detection was important. And then what do you do with the lines? Well, Hough came up with this thing, and-- this one idea. And this is probably-- as far as I know, this is the first machine vision pattern. I think this was submitted in 1960 and issued in 1962.

And it's pretty short. I don't remember whether I loaded it up on Stellar. It's not particularly interesting, but we'll go through it. And at that time, there was quite a bit of negative feeling in the machine vision community, which was, I don't know, a dozen people at that time. You know, why pattern this thing?

Anyway, he patterned it. So what is it? And again, the context is we're trying to see lines in photographs of Wilson bubble chamber pictures. And so he's looking for possible lines. And you say, well, just use the edge detection methods we've talked about already.

Well, the trouble is that these lines were little dotted lines. They were little bubbles. And the bubbles were not spaced equally, and they were not all the same size. So it wasn't clean. And also, because they weren't edges, they weren't transitions from dark to bright. They were transitions from something to something and then back again. But that would have been not too hard to deal with.

So he came up with this idea that we map from image space. so here's our image space. And now in this image space, we can have all kinds of lines. And we map to a parameter space four lines. So you know how many numbers do we need to describe a line? Well, we already did that.

So one way, which I'm not too excited about, but there it is, is y equals mx c. So it apparently takes two numbers to describe a line. And then we-- did I use m here? Let's see here. OK. And so if I have a line in this space, that maps into a point in this space.

And so if I have some local evidence for a piece of one of these bubble tracks, maybe I can map that into this space and accumulate evidence for different possible lines. And so like, that fragment might go there. And then this fragment, that's-- I can fit a line to that. And then hopefully, that'll come to near the same place, and so on.

So the whole idea is that, then, to have an accumulated array and count the evidence for each of the possible parameter combinations, and then look for peaks. And those will correspond to lines in the image. And this can be generalized to other shapes, but let's stick with lines.

So here's another thing. So suppose that I have a line in this space. What does that correspond to in the original image space? Well, it turns out because this equation is kind of symmetric-- if you write it the right way. Let's rewrite it. I can write the equation of the line this way or that way.

And then it's clear that really, those two spaces are symmetric. They're complementary, mapping from one to the other. And so what does that mean? Well, that means that a line in this space on the right corresponds to a point in the space on the left. So that corresponds to, let's say, that.

And so what use is that? Well, that's pretty exciting, because I may not have a good estimate of the line at all. Maybe all I've got is a bubble. I don't really know what it is in the larger scheme of things.

Well, suppose this is my bubble. What does it tell me? It does tell-- it doesn't tell me what the line is. But it's one of those things, where it constrains the number of lines. So all of the lines that this could give evidence will go through that point. And those are all of the lines along here.

OK, so that's an important insight, that if I have a nice clean edge. I can fit a line and I'm done. Well, if I have a bubble chamber picture, I just have these bubbles. But each bubble gives me evidence about a possible line. I don't know exactly what it is. I've reduced the problem from a 2D unknown to 1D. Should sound familiar.

And so what do I do? Well, I just take every bubble, and I find it's transform in this space, and I use that to accumulate some total. So there's a bubble. Now I'm going to get another bubble. And maybe that will give me that line. And then I'll get, I don't know, another bubble, and that'll give me this line, and so on.

And you can see how each of these bubbles contribute some evidence. And then if I keep track of that by having accumulators over here, I can look for the peaks in this accumulator array. And they will correspond to bubbles that line up along some line. So that's the basic idea of transform.

And the key idea is that there's this mapping, and that it's symmetric. Let me just actually do an example. Oh, let's see. How does this work?

So here's my image. And I have a line here. This was supposed to be 0, 1, and this is minus 1, 0. And this point would be minus 1/2, 1/2. And let's put one more in. Four is 1 comma 2.

OK. So let's suppose we have bubbles at those points. Then we go to the transform space. And let's take number 1. So number 1 says x is 0 and y is 1. So that's this one, point number 1.

And if I write that into the equation, y equals mx plus c, that means that c is 1, So then I can plot that in here, and I-- do I have it reversed? m going up, yeah. OK, c equals 1 in this space is that line-- oops. That's the line c equals 1.

So this point gives me evidence that the line I'm looking for is one of those lines. Well, each point here corresponds to a line in that space. OK. Now suppose I pick another one. So let me take point 2, x equals minus 1, y equals 0. That's this one, point 2.

And if I plug that into y equals mx plus c, I get m equals c. And so that-- so this is line number 1. So that line looks like this. This is line number 1. And so I'll go into the accumulator ray, and all of these cells will be incremented.

Let's just do one more. Let's do this one, point 3. So for 3, we have x is 1, and y is 2. And for that, the equation is m equals 2 minus c. So that line, 2 minus c. So that'll start at 2 and then go negative, go like that. So that's line 3.

And you can see what's happening, which is that there's one accumulator that will be repeatedly updated. Now, in the presence of noise they won't be all perfect. And instead of just always hitting that accumulator, we'll be hitting neighboring cells a little bit.

But the idea is that the cell that has the right parameter in it will be incremental the most. And you can do more sophisticated things. But the key idea is that we have a space for the possible parameters of the transformation. We have each point in here corresponds to a particular line. And then, by the way, each line in here corresponds to a point. And so we can gather up the evidence.

And even if the line is really ratty, just these bubbles distributed in unequal intervals, we can get a large count there. So here it's used in line detection, but it could be equally used in edge detection. And it was in some cases.

It's not used much in edge detection-- in line detection now, because we have-- if the image is reasonable, we have much better methods. Their problem was that at high magnification they just had these isolated bubbles that weren't quite on the line and were different-- and so this is a good way of trying to deal with that noise, if you like. So.

Absolutely. You've just invented the extended Gauss Hough transform. So yes. So that's definitely a way to go. And they're going to be obviously trade-offs. Like, if you have something that has a lot of parameters, that space gets large. And then you have to deal with the cost of storing the quantized version of it.

And there are little tricky things like, if you divide up the space, you quantize it into little boxes. If you make a tiny change in the data, you may go from one cell to the neighboring cell. And what do you do about that? So there are tricks for dealing with that.

But let me do another simple example, which is circles. And let's suppose we know the radius of the circle. So we're looking for, again, is just x and y, just two parameters, center of the circle. So one way to introduce this story is that in LTE-- Long-Term Evolution-- in cell phones, the signal from your cell phone arrives at a tower at a time after you sent it, of course, by an amount which depends on how far away you are.

And LTE, unlike CDMA, uses time division multiplexing. So you get a slot to send your data, and somebody else gets a slot to send their data, and somebody else gets a slot to send their data. And if they don't fit in that slot, they're going to interfere with each other. So it's very important that your phone use a timing advance-- in other words, that it knows my signal is going to take 5 microseconds to get to the tower, so I have to send it 5 microseconds before the appointed beginning of the time slot.

S actually, it's even worse than that, because it doesn't know the time. I mean, it's got its own internal clock, but it's not accurate to nanosecond accuracy. There's a different clock in the cell tower which is. But so how does this work?

Well, the cell tower sends out a signal, and that arrives at the phone. And then the phone sends something back. So it is possible to get the round trip time, even if the two clocks are not synchronized. So details we'll leave out, but basically, you can get the timing advance, which is what the cell phone needs to use so it correctly inserts its message into the string of messages.

And that, then, obviously tells you how far away you are. So in Android, you can write an app which-- well, apparently with chalk on my fingertip, the fingertip sensor doesn't work. Anyway, we can get the timing advance, and then you can do various games with it.

One of them is if you know where cell towers are, and you have the timing advance from more than one of them, you can determine where you are. Or you can turn it around and say, I don't know where the cell towers are, but I'd like to know where they are, and you wander around. And you measure the distance from various places that you know from, say, GPS where you are.

So there are problems of that type which involve circles. So let's see how we might use an extension of the Hough transform to do that. And keep in mind that we know the radios. So the timing advance gives us the radius. So that makes it a little simpler. We'll talk about the more general case in a second.

OK, so I'm here and my cell phone told me that the timing advance was a microsecond. So the tower is, what, 300 meters away. And actually, the quantization isn't very good. It's like 150-meter increments. But anyway.

So that means that I don't know where the tower is, but I know it's on the circle. And that's just the same thing we've been doing all along, that we get a measurement. It doesn't give us the answer, but it constrains the answer. And then we take another measurement.

So now you can imagine that we could move around. And say now we're here, and we get a different radius. Well, it's not a-- move it there. So that's radius 1.

And you'll say, oh, well, obviously the answer is here and there. Well, yeah, OK. That's assuming that these measurements very accurately, which you don't. And it leaves you with a two-way ambiguity.

So a better way is to just keep on going. So we've got x2, y2, draw a circle based on that radius. And in this case, they don't intersect so something's gone wrong, which can happen. And in practice, you would have a situation like this.

So that's a Hough transform-like method. All we need to do is have an accumulator array that covers the possible geographic position. And then we use this method to update it. So we reset it all to 0. The first measurement we get, we go around the circle, and we increment all of the accumulators we hit on that circle. Next measurement, we go around that circle. And hopefully, the true location of the cell tower will then be the one where there's a very large accumulated total.

So that's a simple generalization. And a more interesting generalization, you know someone mentioned higher-order polynomials, and that's certainly a possibility. In that direction, what if we don't know the radius? Suppose we have a circle searching-- this is now a different problem. We're searching for--

So there's a circle with the center x0, y0, and radius x0. And we get these measurements. Now, the parameter space is larger. So over here, conveniently the parameter space was two dimensional, just as the theta space.

Well, here we're going to have a three-dimensional parameter space. And we can set up an accumulator right here. And each point in this space corresponds to a circle at a particular position in the plane and a particular radius. And so we can just collect up the evidence for it.

And every time we find a point that is on the circle, we update all of the accumulators in a cone, because, OK, it could be zero radius, meaning it's right where I am, or it could be slightly larger radius and it's on a small circle, or it could be a big radius and on a big circle. I don't know. But now-- and again, I don't get the answer from that single measurement-- I do this again and again, and I get a bunch of cones.

And they all intersect in some complicated way, and, you know, I don't really care. All I care about is that the answer is going to have a lot of contributions. A lot of these cones-- well, ideally all of the cones will go through the answer with noise-- most of them will, not all of them. And so that's kind of the idea of the generalized Hough transform, that we can pick some other shape.

And just the key idea is that we have the original space, and we have the parameter space. And we collect evidence, just as in the pattern, and we accumulate the evidence. And we have a score surface. And we find the peak of the score surface. And if it's above some threshold, then we accept that as the answer.

Think about the r equals 0 plane. Well, if r equals 0, then I'm there. It's a circle of zero radius. So I can just contribute to this cell. But now suppose that r has some non-zero value. Well, that means that I'm not at the correct answer, but I'm higher up in here.

And the possible locations for the-- so I'm here. And the possible locations are on the circle. And then when I change the radius, I'm going to get larger and larger circles. And these are the sections of that.

So the parameter space can get complicated. These are simple examples. And if it's a high-dimensional problem, the parameter space can be expensive to maintain and, in the presence of noise, may not work that well.

So Hough transform is typically not used in its normal, originally defined mode to find bubble chamber tracks or lines or edges. But the idea of the Hough transform is used often as like a subroutine of something more advanced. And the key idea, again, is just that we have this parameter space. And we quantize it, and we accumulate evidence in it. And it doesn't need to be about edges or something like that.

OK. Now, we didn't say much about it in this pattern, but it's important part of the pattern. And right in figure 1, it started off by saying, basically, low-pass filter sample, subsample. So let's talk a little bit about sampling, subsampling, low-pass filtering. And that was part of the working at multiple scales.

Now, there are different motivations for working at multiple scales. One is that you can reduce the computation by working at lower resolution. So if you can get the result you want at lower resolution, it makes a lot of sense to do so.

Another reason is that sometimes, features like edges or texture are apparent at one particular resolution level, but they're not so obvious at other resolution levels. So we think of an edge as a kind of ideal step edge. But we saw, for example, in defocusing, it won't be an ideal step edge. And there will be a scale at which it's the most apparent. And at other scales, it might be such a slow, smooth transition that it's hard to detect.

So let's talk about multiple scales and subsampling. So we already hinted at this, that it's often not that costly. So let's suppose we reduce the number of columns and the number of rows by a multiplication by a factor r, where r could be 1/2. But let's keep a general.

So then the amount of work is going to go as work, and also the amount of space if you keep things around, nm plus r squared nm plus, r to the fourth nm, et cetera. So the overall work is larger than what we have for just the higher-resolution image but not very.

So let's look at r equals 1/2. I think we already looked at that case. Then we get 4 over 3 nm. so in that case, it's just like 30% more work. It's not huge.

How about r equals 1 over square root of 2? Then we get 1 minus 1/2 is a 1/2. And we flip that, we get 2nm. So that's more work. But this turns out to be important.

So we're-- the Hough is the obvious thing. We're just going to take, for example, 2-by-2 blocks and take the average, and that's the new value. And we find that that's actually not a very good way of doing it. But conceptually, we can think of that as a very simple subsampling method.

And so why would we go to something else? Well, it turns out that's very aggressive. And you can find serious aliasing problems when you squish things down that much. So it might be better not to be as aggressive. And so 1 over square root of 2 is halfway there.

And that one is used. So how do you do that? Well, here's one way. You know, what we're going to do is basically reduce the number of cells by 2. So over here, we reduce the number of cells by a factor of 4 going from the 2-by-2 blocks to the 1.

So can you think of a way of subsampling this grid so that you reduce the number of cells by 2 rather than 4? Anyone play chess or draughts? And so a checkerboard will do it.

So we can do a red, black or red, white-- red, black. So if we just retain one of the colors, one of the two colors, by definition, then, we've halved the number of cells. And it's probably not the first thing that came to your mind, because it doesn't look like it's a nice rectangular grid.

But it is, because all we need to do is think of it as a grid running in that direction. So because these cells follow a regular pattern here, and then the next row is here, and same in this direction. So if we're willing to deal with a 45-degree rotation, this gives us a square root of 2 reduction.

And the other thing it does is it increases the spacing by square root of 2. So in this case, the spacing increased by a factor of 2. So now the spacing between neighboring cells is multiplied by the square root of 2. And so that's a little odd 45 degrees. But think of it. You do it the next time, you're back in sync with the original.

So on a related point, there's a method called SIFT, which is used for finding corresponding points in different images of the same scene. And it's used widely to patch together multiple images to produce 3D information about, for example, popular sites that tourists go to where you have thousands of pictures. And you are trying to take two pictures taken from two different positions, with different cameras, under different lighting conditions, and you're trying to match them up.

There's this method due to Lowe called SIFT, which gets descriptors of points in the image that are attempting to be as unique as possible. And he uses much smaller-- much less aggressive. So he uses multiple steps per octave.

So we've gone down a whole octave in one step, you know, a factor of 2. Here we go down a half an octave. It takes two steps to go down a whole octave.

And I forget what he recommends in his pattern. It's, I don't know, six or something. So it's more like a musical scale, where an octave is divided into-- eight nodes? Anyone know? OK. And so they're not equally spaced, but that's the idea.

So yes, it might seem a little odd that we're not going so aggressive with a factor of 2. But actually, there are good reasons not to always do that. And, you know, SIFT is a classic example of a situation where we do not subsample as aggressively. But we're out of time, so sorry.

OK. I've got proposals for those of you in 6.866, of almost everyone. If you're one of those people who hasn't yet sent in a proposal, please send it in. And yesterday, I opened a cookie, and it said, slaying the dragon of delay is no sport for the long winded. So please take that to heart if you're one of the people who has not sent this proposal in yet.