[SQUEAKING]

[RUSTLING]

[CLICKING]

PROFESSOR: We started off talking about the two aspects of image formation, where and how bright. And where we talked about perspective projections. And in a camera-centric coordinate system, it's very easy to just get with that, and the extent of that to be able to talk about motion. And so we just differentiated that.

And we had a slightly different version of this where we made use of the perspective projection equation in order to write it in. Then we introduced the idea of the focus of expansion. That is the place where u is 0. And clearly, that's where this part is 0. And so if you solve for x, we get x over there. It's u/w.

So the focus of the expansion is the point in the image towards which you're moving. And then we introduce the [INAUDIBLE] and talked about various ways of estimating that. Then from a somewhat different point of view, we looked at the image solid. So we're thinking about an image as-- brightness as a function of x and y, and sometimes x and y and t. And so there's an image solid video.

And in this case, we looked at the possibility that the brightness of an image of some point in the environment doesn't change with time. So we introduced the constant brightness assumption. So as we follow some point and its image in successive frames, we are saying that in many circumstances, the brightness won't change. And we can exploit that.

So if we-- it's quite interesting to look at this image solid and slice it in different directions. And you'll see kind of a very streaky nature because of this phenomenon. So the slices, of course, are not independent. And we see that things are streaky, like extruding toothpaste with multiple different colors in it as things move in the image.

And then from this, we got the brightness change constraint equation, which gives us a relationship between the movement in the image and the brightness gradient and the time rate of change of brightness. And we then addressed the problem that this is not giving us the ability to solve locally for velocity, because this is a linear equation in u and v. So it just defines a line in velocity space.

So looking at a single pixel, we can't recover the motion, unlike the 1D case, where we could. And so we need more constraints. Well, a very extreme form of constraint is where everything is moving at the same speed. And so as I announced, there's a paper under Materials on Stella that goes into that. And it's doing the last part of the previous homework problem.

And what it does is minimize some errors. So if this applies at every pixel and we have a constant u and v for the whole image, as in the optical mouse case, then we can write the problem this way. And again, that integral should be small, or it should be 0 if there was no error in u and v, and if there was no noise. But we're satisfied just to minimize that. And that's going to be our best estimate of u and v.

And it's highly overconstrained. We've got one of these equations for every pixel. And we're only looking for two unknowns. So this is a case where we have millions of equations. And we've only got two unknowns. So that's very favorable. That means that the result is going to be much more accurate and reliable than it would otherwise be.

So doing that, we obtain the linear equation in the unknowns with a symmetric 2 by 2 coefficient matrix. So we just run through the image, and we estimate the gradient Ex Ey. We estimate the rate of change of brightness, Et. And then we just accumulate these totals. And when we're done doing that, we have two linear equations in u and v. And we all know how to solve linear equations, particularly if there are only two of them.

Now, as usual, we need to look at when that fails. And so we did that. We said that, well, it depends on the determinant. So the problem is when that coefficient matrix is singular. So we have a problem if that is 0, or if that's equal to that. And we looked at various conditions, like e equals 0. Of course, if you have a black image, then that won't work. And also, if Ex is 0 or Ey is 0 as well.

And just let's look at one more. Suppose that we have an image like this. And what we're trying to do is figure out whether this motion recovery is going to work. So how can we attack that? Well, in two ways, the one is what type of an image is that. Can we intuitively see why that's going to work or not work? And the other one is just to break down and compute the derivatives.

So e sub x is going to be the derivative of this thing plus the derivative of the argument with respect to x many times. And so the derivative of this with respect to x, of course, is just a. And take out the x so it doesn't look confusingly like a times.

And then I can look at the y derivative. So this is a very particular type of image. And in this case, I have Ex and Ey. f may be some complicated function. But the important thing is that Ex and Ey are in the same ratio everywhere. And so in this integration, I can replace Ey by b of a times Ex. And so this is going to be true. This condition will be true, and it'll fail.

So this is another way of saying the condition under which this method won't work. And then we can look at what kind of an image is that. Well, as usual, I can't draw gray levels on the blackboard, and not draw gradients. But I can draw contours of constant brightness, isophotes, which are perpendicular to the gradient.

And so what are the isophotes? Well, the isophotes are where E x, y is constant. That means where f of ax plus by is constant. That means where ax plus by is a constant. And that's the equation of what? It's a straight line, right? So the isophotes are straight lines. Also, they're all parallel straight lines. They only differ in C. a and b are fixed ahead of time.

So this is the kind of image that gives us trouble. Yeah, we know that, right? Because if it slides in this direction, we can't measure that. There's no change in the image. If it slides in that direction, we can. But it doesn't allow us to determine the other part of it. So then that's the optical mouse problem.

So from that we went to time to contact. And we looked at-- so we had this. There are various ways of rewriting that. Let's see. Which way around do I want to do this? So w is the z component of the motion in the world. So that's dZ dt. And I don't know. That may or may not ring a bell. But that's a derivative of log of z.

So if I plot things on a logarithmic scale, this is just the slope of the graph on that logarithmic scale. So that's interesting. So it's all dependent on ratios, fractional parts, rather than absolute values. So what's important is by what fractional part does z change in a certain time interval? Not by how many meters. And that's one reason why we could do this without calibration. We didn't have to know what the focal length of the camera is, for example, because it's only the ratio, the fractional part that matters.

And now another way to think about time to contact is in terms of image size. So suppose here is some object in the world of size S, and here's its image of size little s. Then I can write an equation relating those quantities based on the triangle, similar triangles, this triangle on the outside, and this triangle in the camera. So I've got s/f is S/Z. It's just the lateral magnification of the camera, which in this case is much smaller than 1. The image is much smaller than the object. But we call it magnification anyway.

Well, I can cross multiply. So I have that relationship. And why do I do that? Well, because now I'm going to differentiate that and see how it changes with time. And so s is changing. If we're approaching the object, then s will be increasing. So s will be changing. So we're going to have Z ds dt. And it's a product. So we also get Z. Get the other one. We're going to get s times dZ dt.

And then we need the derivative of this product. Well, the size of the object, presumably, is constant. We aren't changing the imaging parameters. So this is the derivative of that is 0. And the derivative of 0, of course, is 0. So we get this relationship. And so this tells us that-- which way around did we use it over here?

S ds dt over S is Z over dZ dt over Z. So the change in image size, in fractional change in image size is exactly the fractional change in distance. And so for example, if the picture of the image of the bus increases by 1% as you go from one frame to the next in your video sequence, then that implies that the TTC is 100.

And so that's-- at 20 frames a second, that means it's only 5 seconds away. So one conclusion is that in a lot of important practical cases, the time to contact is not tens of frames, but hundreds of thousands of frames. Otherwise, you'd have a problem. You'd probably be just about ready to crash into something and may have trouble compensating.

So if the time to contact in many cases is thousands, that means that the fractional change per frame is 1 over thousands. And so the fractional change in the image from frame to frame is relatively small. And so that means if we were to use a method that's dependent on actually measuring the image size, estimating how big the picture of the bus is in your image, it better be really accurate, like one part in several thousand.

And that means it's going to need subpixel accuracy, that it won't be good enough to simply measure where the front and the back of the bus is in the image. And that's why that turns out to be not a good way to estimate the time to contact, as opposed to the method that we described. So that's one thing I wanted to get across, this very simple relationship that if there's a certain percentage change in size between frames, that translates directly into a certain percentage change in the distance. And that directly translates into time to contact. So it's a very easy way to understand that.

Now, when we did this, or you did it, we had a very simple situation. We started off moving directly towards a wall so that we had constraints on both the direction of motion and on the surface we're looking at. So the very-- you may remember the very first thing we calculated was C, which was the component in the Z direction. And we had some simple ratio of two integrals.

And that's the case where we're moving straight towards the wall, the wall is-- what does it mean? The optical axis is perpendicular to the wall. Z is constant on the wall. It doesn't vary as we go left and right. So that's a very simple case.

So then we said, well, let's be slightly more general. Let's assume that we could also be moving sideways as we're going along. And then we added the motion in x and the motion in y. And we had the slightly more interesting problem where we were looking for three unknowns, A, B, and C. And we ended up with three linear equations and three unknowns.

And right now I remember what I was going to say. Again, this is all spelled out in the paper that's on the website on time to contact. I guess the full title is "The Time to Contract Relative to a Planar Surface." And now the paper discusses some other things as well. It's just like the other paper. The first half is exactly what we did in class, and then it goes off into some other directions. Same here.

So is this the most general we can get? Well, we're still making an assumption that Z is constant. So we're approaching the wall, and the optical axis is perpendicular to the wall. Well, what if our camera is tilted, or conversely, we're approaching a wall that that's tilted in the world? So a different generalization has Z be a tilted plane so that it's no longer the case that the depth is constant as I scan left and right or up and down in the image.

And well, what's the equation of a plane? Well, it's going to be some linear equation in x and y. So one way I could write it is in that form. So this could be a more complicated model to look at. And well, you might expect at some point the equations get pretty complicated. And in fact, there might not be a closed form solution. And that's fine. You can do it numerically.

But in terms of understanding what's going on and the noise effect, it's nice to focus first on cases where there is a closed form solution. So what you can do, actually, is say, fine. So now we will generalize it by allowing the plane to be tilted. But let's go back to the case where we were moving straight down the barrel, straight down the optical axis. And in that case, then we'll only have these three unknowns. So instead of having A, B, and C be the unknowns, we have these three or some function of them.

And so we want to do that because it's pretty straightforward, more messy algebra. We end up with three linear equations and three unknowns. And we can solve for that. So we can do the time to contact in the case where the surface is tilted. So we're no longer making the assumption that we're driving into the side of the truck coming out of a parking lot. But we could be coming at an angle. So that's an interesting case to consider.

What if we do both? What if we allow the surface to be tilted as well as our motion to be general? Well, we can formulate that problem pretty easily. And we end up with six unknowns. Unfortunately, they are no longer linear equations. And so from the point of view of pain and agony, they're not much fun to write down. And also, it's unsatisfying because we end up with this mixed set of equations. Some of them are linear. Some of them are quadratic. And it's very hard to say anything general about them. Whereas in these special cases, we can do a full analysis of noise and so on. So we're not going to do that. Just you know that it's there.

Then we get to why is the surface planar? Well, the surface is planar because it's going to give us linear equations. So we start with a planar surface. Now, real surfaces may not be planar. And then what? Well, we can approximate them by polynomials, some locally quadratic surface. And we go through the same process. We set up a least squares problem. And we, unfortunately, won't find closed form solutions. But we can set it up so that some numerical process gives us a solution.

Now, the reason we're not doing that is mostly because it actually doesn't buy you anything. So in practice when you implement this, you find that modeling the surface as planar gives you a very good estimate of the time to contact. And if you model it as something more complicated, now you have more unknowns, which is good in a way, because it allows you to model the world more accurately.

But at the same time, you lose that overconstraintness. Every time you introduce more variables, there is an opportunity for the solution to squiggle off in another direction. So there are some pluses and minuses. And overall, the only time you want to even think about that is if the object has a shape in depth where the depth change is similar to the distance from the object.

So if the truck is over there-- I'm 50 meters away from the truck, and the side of the truck is-- one side is maybe 2 meters closer to me than the other, it makes no difference. I mean, it's a 4% change in distance. And that won't affect anything. If I am right in front of the truck, I'm 2 meters away and one side is 1 meter and the other one is 4 meters, then yes, then you may need this. But we found in practice that we don't need that extra level of sophistication. These two things are sufficient in practice.

Then let me just briefly talk about multi-scale. So when I showed you the implementation, when you got really close to impact, things kind of just fell apart. So the graph of time to contact estimated was quite similar to the actual time to contact. So this was a contrived situation with constant velocity so that the time to contact decreased linearly as time went on, because we got closer and closer to the surface.

And then when we looked at the computed results, they were something like that, noisy. And that's partly because the measurement of the position was not very accurate. It was eyeballing down on the measuring tape. There was an interesting offset vertically, so there's a bias. So it's not just noise. Actually, the estimated time to contact was overestimated, which in itself isn't good, because if you're about to crash into something, you don't want to be told that actually it'll take longer than the true time.

On the other hand, since it's a systematic fixed bias, you can compensate for it. It doesn't mean that you shouldn't try and figure out where it comes from. But it's pretty simple to just fit a different slope to this. But then at the end here, we had some spikes. And basically, the results were not reliable. And we already mentioned some reasons for that.

And one of them is that the image motion is large. So earlier we said that when the bus is far away, the image motion is very small. The image of the bus will tend to expand and contract and move by a fraction of a pixel. And that's where these methods really excel. As we go along, we've been making some assumptions that certain distances, the epsilons are small. When we estimate E sub x, E sub y, E sub t, for example, we're taking a finite difference and saying, oh, this is almost a derivative, because epsilon is small.

Well, that won't work if we have a large jump in x, y, or t. And so that's one reason that this falls apart. I mean, there are other reasons. One of them was that the camera went out of focus. And so you didn't have a clear picture of the object anymore. But this first part is easy to deal with. As I already mentioned before, I just want to reiterate that if we have an image with less resolution-- say we have half the number of rows and half the number of columns in the image-- well, then the motion in terms of pixels per frame is half what it was before. And so what was a large motion in the original raw image is now half of that.

And so that means that you'll still have it falling apart, but it will fall apart later. So this part will still be OK. And then it'll fall apart down there. And of course, then you can repeat that process, and say, OK, so now in that image suddenly the motion has gotten to be more than a pixel per frame. So let's, again, subsample, average and subsample. And then we can continue it. And so multi-scale just means that we work in that set of images that become smaller and smaller. And we can handle motions that become quite large.

And also, I mentioned that if we do the simple 2 by 2 block averaging, that means the second image is only a quarter of the size of the first one. So the amount of work is 1 plus 1/4 times what it would have taken just on the raw image. And then we do it again, so that's going to be a 16th. And so the total amount of work-- well, writing the code, of course, takes a bit more effort. But in terms of the time, it's not a big penalty to work at multiple scales. And you get hugely improved results.

And we'll talk a little bit later about how to do the subsampling. Those of you who've taken 6003, of course, realize that you can't sample without getting aliasing unless you've low pass filtered first. So actually, what you want to do is low pass filter, and then subsample. And you don't necessarily need to subsample on the scale in x by 2 and y by 2. You could subsample by-- I don't know-- square root of 2, which is less aggressive and introduces fewer artifacts.

But for the moment we'll ignore that and just take the very simple idea of 2 by 2 block averaging, which is a crude form of low pass filtering. And it doesn't do exactly what you need to do, but it removes-- it suppresses some of the high frequency content. And while there will be aliasing artifacts, they'll be greatly reduced. And that's such a simple method to implement. And it works pretty well.

Well, let's talk a little bit about what to do-- what do I do with time to contact? So there's a number of interesting applications. Every year there are several incidents with airplanes on the runway where wingtips are taken off. So these planes have very long wings, and they're typically swept back. So they're not terribly visible. And so there's an opportunity to bang them into a building or into another plane. And that's a really expensive thing to deal with. It's not life threatening in most cases, but it's something that you try and avoid.

And as you approach the place that you get on the plane, there's often some person down there with some red lighted stick. And they're called wingmen. And the reason is they walk under the tip of the wing so that the pilot can look back and see where's the ground projection of the wing. And I'll try not to hit that staircase with the wing. So it looks like you could implement time to contract. I mean, you can implement it easily on Android, for example.

So you could build a really cheap little box. The only purpose in life for it is to look out and see if something is rapidly approaching, and then give you a warning. And so we've thought about suggesting that to airplane manufacturers and such. And it didn't get anywhere. But Boeing took the idea, and they came up with a $150,000 radar solution. And that's obviously going to be much more fun for the corporation to implement than something as silly as this. So anyway, sour grapes.

The next project is NASA landing on Europa. So Europa is a long way away. And we have some idea of what's on there, but not a whole lot. So it's not like we have detailed topographic maps and imagery. And so they want something that reliably brings down a spacecraft. And so one idea is to use time to contact in control. So let's look at how we do that.

So we have a typical control system. We input some desired time to contact. Then we have an actual estimated, and we subtract the two. And that gives us some kind of error signal. And we multiply that by a gain, and we use that to control the jet, the rocket engine to change the acceleration.

And then there's a dynamical system, which is second order in the sense that we're controlling acceleration, not height directly. Height is two integrals down from the part that we can control. And then there's an imaging system. And so this system does something very simple, which may not be the best you can do, but it's easy to analyze, which is to try and maintain the time to contact the same.

So if your measurement says that at the current rate of descent you're going to have a shorter time to contact than desired, then it'll add some more force to the engine. And conversely, if it looks like you're kind of hovering too much, you should be dropping down, then the time to contact will appear large. And then the error signal will cause you to reduce the engine output. So a very simple system.

And we saw that time to contact is very easy to implement. And it doesn't care what the imagery is to a large extent. So we don't really know what the surface of Europa looks like, except from far away. But we're not depending on some particular texture, or calibration, or topographic map of the surface or something. This method works with any texture. Well, except we saw that there were certain special textures where it would fail. But presumably, Europa hasn't been painted in one of these unique stripy patterns.

What are the dynamics of this? Because this idea of time to contact control can be used in other situations as well, such as in autonomous cars. But let's focus on the descent here. Now, why constant time to contact? Well, we don't really know how to accurately, reliably, easily compute height from a monocular camera image, unless we have some target, like there's a Walmart, and we know what size Walmarts are. So we can compute how high we are. Well, there aren't any on Europa. I hope so. That won't work for us.

So if we could separately compute height and speed, then we could do much more sophisticated things. But we know that we can very robustly get their ratio in a very simple way. So that's the attraction there. So we've got Z/w is T. And now we are assuming that T is constant.

It's very curious that when DARPA had the grand challenge, of course, MIT was involved in that. And we instrumented a car. And there was one sequence that we thought would be interesting to compute after the fact. They've recorded all this video. Let's do something with it. Let's compute time to contact.

And so there's the vehicle coming out of a parking lot, and it's approaching the MIT transport little bus. And if you plot the time to contact, it's almost constant. So the vehicle is slowing down. The closer it gets to the bus, the more it slows down. And so I have no idea what in the control algorithm of that autonomous vehicle did that, but it was interesting to observe that it used a constant time to contact control to approach the bus without running into it.

Well, here's an equation that we can solve. So it goes Z over dZ dt is T. And so dZ dt is 1/T Z. And well, there's a ordinary differential equation that you should know the solution for, keeping in mind that T is a constant. So dZ dt is proportional to Z. What sort of function does that? Quadratic? Exponential? Anyone?

OK. So we get Z is e to the minus t over T, something like that. We differentiate that, and the derivative is proportional to the function itself. And so if we implement this constant time to contact control system, what we'll find is that we're going to get a descent that looks like this. So here's our Z0. So it's going to be nice and gradual and smooth, and it will never get there. So that's the downside.

So what do you do there? Well, we can do what flies do. They use constant time to contact when landing on the ceiling. And when their legs touch the ceiling, they stop. So we can have wire hanging down from our spacecraft, and when it touches the ground, we shut off the engines. And then it'll just fall those last meter or 2. And that method, of course, has been used in planetary spacecraft before. But we can just combine it here with the time to contact.

So at a certain point, we do have a distance measurer, an actual wire, and then it just drops under gravitational control until it hits the surface, I guess. So that's one aspect of this. And we can go into an error analysis of what will happen there.

Just for fun we can compare this with a more traditional approach. So a more traditional approach would be we run the engine at its rate of speed. And we decelerate. And we only turn it on at the time where we need to turn it on so that we don't crash into the surface. So now that method requires that we know the distance to the surface and we know the velocity. So we have to know a lot of stuff, where with the time to contact control, we don't. You just keep the time to contact constant.

So one advantage of this alternate method is it's more energy efficient. So here we'll be kind of going almost into a hovering mode. So we're kind of wasting fuel in a way. So there's a trade-off. So we're ignoring the fact that the spacecraft is getting lighter as it's using fuel, and so the acceleration will change. We'll just assume that it's constant.

And so we integrate that once, and we get a constant of integration. And we can apply the boundary conditions, which are that at some point we reach the surface. And then we integrate a second time. We integrate that, and we get Z is 1/2 a t squared plus some other constant of integration. And again, we use the boundary condition, and we end up with z is 1/2 a. Of course, you can do this the other way around. It might be easier to start off with that, and then just differentiate to get to the constant acceleration.

So why did I even bother doing this? Well, because it's interesting to ask, under that type of control, what is the time to contact? And so how do we compute that? Well, we need Z and dZ dt. So we just take the ratio of these two. So T is-- and so that's going to be 1/2. So that's saying what the time to contact is during any part of this maneuver. And it's fascinating that it's a half of what you get over here.

Over here, of course, the time to contact would be T minus T0. But because of this constant acceleration control, we get that result. Anyway, it's just a way of comparing constant time to contact control with a more traditional approach. The more traditional approach is somewhat more fuel efficient. But it's much harder to implement. It requires accurate estimations of distance and velocities.

Anyway, apparently, NASA decided they can accurately estimate distance and velocity. So these crazy machine vision programs, who knows whether they'll work? So they're not going to do that. But you can imagine that there are other applications for this. Because this is a sensor that's really dumb. I mean, it's simply a matter of brute force estimating gradients and accumulating totals, multiplying, and then solving a bunch of equations. So it's very straightforward.

Now, before we go on to another topic, I want to point out another generalization that we'll take up later, and that's optical flow. So the paper on Stella that talks about the optical mouse problem primarily is-- what's the title? "Computational Fixed Flow, Determining Fixed Flow." Fixed flow, what does that mean? Well, it means that the motion of all parts of the image are the same. And as I explained, for an optical mouse, that's a very good model. That's very accurate.

But as I'm walking around the room, the motion of different parts of the image are not the same because you're at different distances. And we saw how we can get that out of a perspective projection equation by differentiating. And we get some terms that are affected by division by z, and so on. And then, also, some of you may be moving around, independent motion. It's not that.

So these problems are particularly easy to solve when there are only a few parameters. So for the optical mouse there are two-- motion in x, motion and y. Well, you might be turning the mouse. So it could be three. And we'll probably take that up in a homework problem. So it'll be a slight generalization of what we did with the added feature that you're not only tracking the position of the mouse, but if the user is turning it, you also want to recover that. Maybe not because that's an interesting input to your GUI, but because it might screw up the estimation of the x and y motion if you do rotate.

Anyway, so those are all cases where we have a fairly small number of unknowns, and we have a hugely overdetermined system. We have some measurement at every pixel. What if we don't have that? Well, that's going to be a problem, because at every pixel we have an equation like that. So we have a constraint. So if we have 10 million pixels, we've got 10 million constraints. Great. Except at every pixel now, we also have an unknown velocity.

Whoa! Cheap chalk. So that means that we have twice as many unknowns as there are equations. Well, that's a prescription for disaster. So highly underconstrained, and it's an ill-posed problem. And in this case, it's ill posed in the sense that there's an infinite number of solutions. And in fact, if you give me a solution, I can easily construct another solution, because all I need to do is at every pixel I need to obey this constraint. So at every pixel, I'm somewhere along this line.

And now you give me the, quote, "correct solution" that says here for that pixel. Well, the thing is that I can go there. It doesn't change anything. And so that's pretty dramatic. That means that I can systematically go through the image, and at every pixel I can give you an infinite number of other values that will work. So we'll need some heavy constraint. And one constraint that we can use and will use later is that neighboring points in the image do not move independently. They may not move at the same velocity, but they tend to move at a similar velocity.

So that's a good thing and a bad thing. I mean, it's good in that, oh, here we've got some sort of constraint. And it's a bad thing, because it's not like an equation that says u plus 3v is 15 or something. It's more vague. It's like, oh, it's varying smoothly. What does that mean? What's smoothly? What epsilon change can you allow? So that makes that problem very interesting and non-trivial. We'll get to that later. We don't have the tools at this point to do that, but it's just to alert you to the fact that the fixed flow isn't the be all and end all. There's more to come.

Now, suppose that you don't have the tools to solve that problem. Well, you can do something, which is divide the image up into pieces, and then apply fix flow to each piece. And the idea is that, well, if we make the pieces small enough, there won't be much variation in the velocity within that piece. And so the assumption that u and v are constant in that little area isn't such a bad one.

And so now there are all kinds of trade-offs, because if we make these boxes very large, we get only a very coarse image of the flow. We only have a flow vector for each of these boxes. On the other hand, if we make the subimage areas very small, we have much less constraint. And we have much less of that wonderful noise suppression property of an overdetermined system.

And also, when we look at small areas of an image, they are much more likely to be similar to the type of image that we talked about that doesn't work. If I look at a very small area of the image and it just has this edge in it, well, that's exactly the kind of thing where the aperture problem comes in. And I can't determine what the motion is in that direction. So yes, you can do this. You can use what we have-- the fixed flow method on a grid of a chopped up image.

But there are trade-offs, and they're somewhat unpleasant. If you make these areas fairly small, some of them may be even more or less uniform in brightness. If I'm looking at that gray wall, it's uniform in brightness. And if it moves, I can't tell. It's just the same. So anyway, that's something that has been done and works, but it's not the solution that we'll be looking for.

We're moving towards talking about brightness more than perspective projection. But I want to just do one more thing with perspective projection. And that has to do with vanishing points. And these play a role in camera calibration or sometimes finding the relative orientation of two coordinate systems.

And just to make that sound less mysterious, if we have man-made objects, they often have planar surfaces. And they often have right angles between their planar surfaces, unless you go over to start there. And so when we look in the images, we're going to find a lot of straight edges often. And often, a lot of them are parallel.

And so we can actually exploit that. So if, for example, you're hovering above some building, like the main buildings at MIT is all on a rectangular grid, you can determine from the image certain vanishing points. And from that, you can determine your rotation relative to the coordinate system of that rectangular block. And so that's something there. Or if you look at a cube or some other calibration object, you may be able to recover parameters of the imaging system. So it's important in the camera calibration, which is, of course, important in robotics and other applications.

So let's just see what this is about. So in the first homework problem, one of the questions you were asked is, what's the prediction of a line? And there are various ways of approaching that algebraically or geometrically. One geometric way is just to say, here's my line. I'm going to connect it to the center of projection. And what do I get? Well, I get a plane.

If I connect-- if I look at the locus of all those lines that connect that point to the line, it forms a plane. And then what is the image of it going to be? Well, I'm going to intersect that with the image plane. So what's the intersection of one plane with another plane? It's a straight line. So there's a simple way of seeing that line in 3D projected into a line in 2D.

It's a funny projection in that if you were to mark this like a measuring tape with equal intervals, and then you look at the projection of those marks, they won't be equally spaced, because the part where the measuring tape is close to you will image with larger magnification so the marks are further apart then. So the fact that a line goes to a line is a little bit overconfident. It's making us think that we understand the problem very well, when actually it's a little subtle.

So the other way is algebraic. And so one way we can do this is to say that a line in 3D can be defined in various ways. One of them is we have a point on the line, and then we have a direction. And we might as well use a unit vector to define the direction. So that's one way of talking about a line in 3D.

What are other ways? Can you think of some other ways? We can do a parametric representation where we have, for example, x is some function of some parameter T, and y is some function of parameter T, and z is some function of parameter T, or we can have some implicit representation, or we can intersect two planes. And why is that convenient? Well, the equation of a plane is a linear equation. And so when you intersect two planes, we're dealing with two linear equations then. We've already seen that having linear equations can be a plus.

But let's stay with this. And in component form, of course, that just means it's x0 plus s. What did I call it, alpha or something? Alpha s. So then we're going to now project that into the image using perspective projection. And we can, of course, use the component form that we used up there, or we can use the vector form.

And so we get-- so unfortunately, this doesn't lead to some very elegant, nice result. Or rather, it doesn't without-- sorry. I guess Z is dependent on gamma. Unless we go quite a long way further than I want to when it becomes nice and elegant again. So that's our transformation. And so s is a parameter that varies along the line. So different points on the line have different values of s. And if that's a unit vector, then s is actually a measure of length along there. Because of that division by Z, it's kind of messy.

But one thing that's interesting to do is to look at what happens when we go very far along the line. So we go there. So we make s very big. Well, that means that we can ignore s0, and we can ignore z0. And we get alpha over gamma. Gamma, not beta. And that is called the vanishing point.

So as you move along this line, you start to go more and more slowly in the image. And you approach, but never reach this point. And so actually the image of an infinitely long line is not an infinitely long line in the image plane. It starts somewhere, and this is where it starts.

Then as we move along the line in 3D, we don't move along the line in 2D in a uniform way, because when we're very far out, we can move a long way in 3D. And it only has a tiny effect in the image. So that's what makes it kind of awkward. Now, one thing that we should immediately recognize is that parallel lines have the same vanishing point, because the offset x0, y0, Z0, the origin of our line, they don't come into this equation. It's only the direction that matters.

And so that's interesting, because that means that if we have a bunch of parallel lines in 3D, they are all going to give rise to the same vanishing point. So if we're looking at a rectangular building, there are three sets of edges, each containing parallel lines. And so we expect to see three vanishing points. So let's see if I can do this. This might be--

Now, this is a bit extreme because I picked the vanishing points pretty close. So here's a cube. Well, not that good. But anyway, if it was for real, there would be three sets of parallel lines, which each give rise to a vanishing point. And so what? Well, the point is that if I can find those vanishing points in the image, I can use them to my advantage to learn something about the geometry of the image taking situation.

And just to let you know that this isn't some vague theoretical thing that nobody cares about, here's an application. So every year a few people are killed on the side of the road because of distracted drivers, even before texting. And so there's an interest in trying to warn whoever's stopped there-- police officer, construction worker, whatever-- that there is a car on a trajectory that may impact them.

So how do you do that? Well, you can stick a camera, maybe an Android phone, in the window of the cruiser. And it's watching-- this is mostly nighttime-- it's watching headlights going by. And it's monocular, so it doesn't have depth information. But it can figure out whether the trajectory is possibly dangerous or not.

But one of the things it needs to do is to figure out the geometry of its coordinate system and the road's coordinate system. You don't want the person who's using it to have to come out with surveying equipment and measure the angles and so on. So the camera has to, on its own, try and determine the rotation of its camera-centric coordinate system relative to the line, to the road, x, y, and z.

And so one way to do that is to look for a vanishing point. So in the case that the road section is straight, you can use image processing methods to find those lines. And then you can intersect them to find the vanishing points. And then you can use those to recover at least two parameters of the transformation. One is how much is the camera turned relative to the direction of the road, and the other one is how much is the camera turned relative to the line to the horizon. So we've got pan and tilt. And we can get pan and tilt using vanishing points.

Ugh, I don't want to do that. So let's think about using this in camera calibration, another application. So in calibrating a camera, there are several parameters you're looking for. The main ones are the center of projection. So you say, well, yeah, the center of projection is where the lens is. Yeah. But where's the lens? And relative to what?

So here's our integrated circuit sensor. And up here is the lens or pinhole for the moment. And presumably, whoever built this thing tries to put the center of projection above the middle of the chip. But that's not necessarily going to be accurately done. I mean, these things are microscopic in some cases, like this thing. It had a focal length of 4 millimeters. And so we're dealing with very small quantities.

Plus, the pixels here may be 10 microns, or in the cell phone maybe only 5. So it's very unlikely that you would be able to position the lens in such a way that it was accurate to one sensor position. So we need to recover that position. And we also need to recover the height of the center projection above the image plane.

And ultimately, we'll be using a camera-centric system that's origin at the center of projection. But to do that, we need to understand how row and column in the image sensor translate into x and y in that camera-centric coordinate system. So the short answer is if you give me a coordinate system in the chip, which is a, let's say, column and row count, I want to know where that point is. And typically, the row and column count won't be taken from the center of the chip, but from one of the corners.

And when you give me the position of the center of projection, what units do I want? The size of the pixel. It'd be nice to have it in microns. But if you don't know the size of the pixels, you can't do that. And conversely, you don't need to know the size of the pixels for projection. We can express that. Like the focal length, we can express it in terms of pixel size, particularly easy if the pixel is square. I mean, otherwise, you've got to deal with the fact that the x and y dimensions aren't on the same scale.

So that's the task. Tell me where that point is. And it may need to be repeated, particularly if it's a camera that has zoom capability, because then all of that's going to change as you zoom in and out. And you'd want to perform this kind of calibration again.

So there are various ways of doing this. One is-- and we'll talk about this some more later. But here's a very simple one. We use a calibration object. And so a calibration object is something with a known shape. And we already talked about that when I was showing the slides, where we used the sphere as a calibration object. So would a sphere be a useful calibration object here?

So the image of the sphere, as you saw in homework problem one, is a conic section. And so for example, if the sphere was directly above on this line, its projection would be a circle. The sphere is up here. And how do I know that? Well, because I connect the sphere to the center of projection. And I get a right circular cone. And I extend that down here and intersect it with this plane, and intersect the right circular cone with a plane perpendicular to its axis. And you get a circle.

But when I move the sphere to the side, it's going to become elliptical. It's going to become-- imagine-- take an extreme case where this is way down here. And now you project it into the image, it becomes a very elongated ellipse. And if you move it down far enough, it'll be a hyperbola. So yeah, you can do that. But it would then require that you accurately determine the position of that figure, whether it's an ellipse, or hyperbola, or circle, or parabola, and its parameters. So that can be done. But the noise gain is high. It's not a very good method.

The big advantage of this is a sphere is easy to make, easy to obtain. A cube isn't. So let's try a cube. I know that a cube isn't easy because when my father got his master's in operating equipment to make objects, his task was to make a 1 centimeter cube. And it had to be accurate to a micron. And it apparently took him quite a while to do it because it had to be accurate to a micron, and the sides had to be at right angles, and so on.

So making a sphere as a calibration object is somewhat easier than making a cube. But the cube has some huge advantages. And one way of exploiting it is this diagram. So if we take an image of the cube, we can detect the edges. We'll talk about that later. And then we can extend them to find the vanishing point. And by the way, the vanishing points don't have to be in the image. It could be that the image you actually see is this thing.

The vanishing points are in that plane, but they, in many cases, are not in the image. And that's why I struggle in part. Aside from having no drawing ability, I struggled with that diagram because in the real situation, they tend to be further out. And so the perspective distortion, as it's called, wouldn't be as extreme.

So what? So I have a calibration object that's a cube. And then I take a picture of it. And I find the vanishing points. What then? Well, one thing I know is that the cube has three sets of parallel line. And those are at right angles to each other. If they weren't, then it would be a parallelepiped, not a cube. So they're at right angles to each other.

And so that's very important because it brings me to the equations for the vanishing points. It means that the directions to the vanishing points are at right angles to each other. So here's my center of projection, and then I have three vectors corresponding to the three sets of lines. So there's one that's going up and down, and one from that side, and one from that side.

And I can draw them through the center of projection. So here's one, here's another one, and here's a third one. And so what are those lines? Well, those are lines in the direction of the 3D lines, which I guess we've lost now. Sc we said that all of the lines in this parallel bundle project into the same vanishing point.

But there's one that's special, which is the one that goes through the center of projection. So think of this whole slew of parallel lines. And they all end up at the same vanishing point. But they can be represented by a single line. We can just pick one. We pick the one that goes through the center of projection. And so that's what this is. So this is-- and then because of the way our projection works, there's a point down here and a point down here in the image plane.

So if you tell me the three vectors that point in 3D along the lines, I can tell you where they will be imaged just by this construction, because here's the direction that's going along one-- here all these parallel lines. They all going in that direction. And all I need to do is follow this backward into the image plane, and that's going to be its vanishing point. So that's vanishing point 1, vanishing point 2. I called them something else. Sorry. I guess I called them a, b, and c.

Now, the neat thing is I know that if my calibration objects is a cube, that these three vectors up here aren't just any old three vectors, but they are all at right angles to one another. Hard to draw that, but there are three relationships between them. So let's call the unknown center of projection p. That's our task. Find p, given a, b, and c. Well, then I can say this.

So why is that? Well, p minus a is the vector along this line. And p minus b is the vector along that line. And those are the same vectors as these two. And we know they're at right angles. And the dot product of two vectors that are at right angles is 0. So we get the calibration, obviously. We take an image. We find the vanishing points. And then we have those three equations, and we have three unknowns, namely the components of p, the center of projection.

Or another way of looking at it is we need to know this height. Call it f. And we need to know where this is on the image sensor. So there are two degrees of freedom here, one degree of freedom there, three total. But simply speaking, we're trying to find where this is. And that's a vector in 3D, so it has three degrees of freedom. We've got three equations. Great.

Well, if you look at them, you'll see that they're second order in p. p is the unknown. And they're second order. They're quadratic. So there's a finite number of solutions, except for pathological cases. But what is the number of solutions? So with a single quadratic, we know that there are possibly two. So maybe there are more solutions.

So there's a thing here called Bézout's theorem, which we'll use quite a bit. And it says that the maximum number of solutions is the product of the order of the equations. So for example, if I have two quadratics, it is possible there might be four solutions, 2 times 2. Here I've got three quadratics, so it's possible there could be eight solutions. So that's unpleasant, so we want to talk some more about that.

By the way, it's called Bézout's theorem after Bézout, who wrote this up, and actually Newton used this result in his *Principia.* But he didn't really formalize it. He just used it like anyone would know this, that kind of thing. And when Bézout wrote it up, he didn't actually rigorously prove it. So there's a whole lot of-- it's as usual. Someone ends up with their name on something, and there's a whole interesting story behind it, like, OK, he didn't-- he wasn't actually the first, or he didn't actually get it right, or something. Anyway, it's an important theorem for us.

Now, with linear equations, the product is always 1, right? 1 to the whatever power is 1. So with linear equations, as long as we can match constraints with unknowns, we're done. Unfortunately, in other cases it's not quite that simple. But we can do something, which is notice that these aren't just any old quadratic equations. They have a very special structure. And we can subtract them pairwise to get rid of the second order term.

So we can get-- so let's see. If we subtract the first and the second, we end up with this. This is 0. And then we can get a few more. So terrific. We've reduced it to three linear equations. And we know that they have only one solution. Well, not so fast. Are those three linear equations linearly independent? We have to worry about that edge case where the matrix is singular, and so on.

Well, the truth is that if we add two of these, we get the third one. So they're not independent equations. So actually, when we get to the third one, we should stop. So yes, we can reduce it to two linear equations. But we're still left with one quadratic. And so the answer is that there are two solutions. And we won't actually do the algebra, but it's pretty straightforward.

I'm not quite done with this. I wanted to say something more, but we're sort out of time. So one thing that I wanted to still say is that those linear equations, what do they define in 3D space? Planes. So each of them defines a plane. And what we're really doing is we're intersecting those planes. And it turns out that two planes intersect in a line. And it turns out in this case that that line contains the third plane. So the third plane doesn't get you anything.

And then just to summarize how this works, and we'll finish this next time, often, we are in need of calibrating a camera. Time to contact is one of the few places where we didn't need a calibrated camera. And to calibrate a camera, we often use calibration objects. I mean, it doesn't have to be a cube. It could be, for example, the corner of a room, as long as you know the geometry of it.

And then in that case, often vanishing points are helpful. And we can work out the geometry of the vanishing points, which lead us to a set of equations. And when we solve them, we find the position of the center of projection.

Now, if lenses were perfect, that would be it. So this would be it for camera calibration. And we'll talk more about camera calibration later. Our lenses have radial distortion, and there are reasons why people don't completely get rid of that. So actually in practice when you do real robotics camera calibration, it's this plus the radial distortion parameter. So OK. That's it for today.