GUEST SPEAKER: Hi, everybody. Today we're going to talk about semantic localization. So first I'm going to talk about what is semantic localization and what is the motivation for it. Then we'll go through an algorithm that will allow us to localize. And then we'll go into how to actually add semantic information into this algorithm.

So our focus what we're coming up with this was the orienteering Grand Challenge. So in orienteering, basically you have a map and you have a compass, and you have to go around to various checkpoints. So as you can imagine, this is sort of difficult, because you don't know where you are on this map except what you can tell from your compass.

And so we weren't sure how to do this either, so we asked some orienteering experts. So basically the key ideas are, if you are disoriented, that's fine. You just need to find a reference point so that you can figure out where you are. You can think about where you last knew where you were, and estimate what your movement has been since you were there. And that can give you some options as to where you are. And you can look around for features that you can identify uniquely on the map.

And this is an example of an orienteering map. If you just have what's over here, it's not very useful. You can't actually tell what those things are. You might guess that green is grass, but you don't actually know what they are. This map isn't useful until you actually have a legend to go along with it so that you can say you have roads, or foot paths, or pits, or different things on the map that you can actually identify.

So this is the difference between how robots typically localize and how humans localize. So humans think about places in terms of features, in terms of what you can do with them, things like that, whereas robots might measure distances with laser scanners to form a perfect map of the entire space, whereas humans might think about obstructions such as rooms and their relative location to each other.

So we can define semantic information as signs and symbols that contain meaningful concepts

for humans. Basically different sorts of abstractions. And why might we want this semantic information? Well, one very important application would be improving human-robot interaction. So say you want a robot that can make coffee. It has to understand commands like, go to the kitchen. Get a mug. Turn on the coffee maker. This is the language that humans think in, and it's very useful if a robot can actually understand this sort of language as well. Know what a coffee maker is, what it does, and where it is.

And additionally, you can get some performance and memory benefits. You're not storing this full map with the distance to every single surface. You're just storing locations of key information. And this means that your search base is a lot smaller too. So you can prune everything that's not related to kitchens or coffee makers, and get a performance that way. Finally, the fact that you can use simply a camera instead of a complicated laser scanner means that this is a lot cheaper and more accessible for robots to have.

So we've talked about what semantic information is. So now let's define what semantic localization is. So basically, it's localizing based on semantic information rather than metric information, like distances. So for our Grand Challenge, we have a map with labeled objects and their coordinates. This is our semantic information. And we want to basically look around and say, what can we see in our field of view? And from that, we want to figure out where could we be on this map.

And it's important to note that map building is actually a very different and hard problem with a lot of other research on it. In our case, we've been given a map, and we're simply talking about the problem of localizing based on that map. And now I'm going to let Matt tell you about particle filters, which is an algorithm for localization.

MATT: All right, so my name is Matt Deyo. Now we're going to talk about particle filters. So this is a tool that we're going to teach you to use, specifically for state estimation, using the given math and measurements. So what is localization? We just went over it. Specifically, it's the question of, where am I? For any system to work, the robot needs to know where it is on a map. This is not so simple of an answer.

Metric localization. As was said before, it is quantitative. So how many meters are you from this wall or from your origin? What's your orientation in degrees? And here's some examples of that. You can convert between coordinate frames really easily if you have metric.

So the localization is, well, in our case, mathematical. The problem statement is, you have a control u. You're going to observe something about your state. It might be a camera image. It might be laser scans. And then essentially on your map, you're going to use probability to figure out where you are.

So this equation specifically is taking into account the probability of being out of position at your current time based off your previous position, the observation that you're taking right now, the command variable you gave it, and then your map, like I said. So that's built on the observation noise, actuation, and then the belief. We're specifically looking at the belief here for the particle filter. So in our case, we're approximating it by several points, also known as particles. So we're going to look at an example here. I have to actually press the Play button. OK. Here's a quick demo. [INAUDIBLE]

So a quick YouTube demo of a particle filter in action. So I'm just going to show it first. So pause. The red dot here is our actual robot position. And it's trying to localize itself in a maze. There's black walls throughout here. And it looks like a grid world, but that initial distribution we had was completely random across all the walls. So it's taking in observations of probably laser range finders.

So it essentially knows that there are walls around it, and it knows that there's not a wall in front of it. So that centers all of these guesses on the center of hallways. Obvious it got rid of things that were close to the walls. There's a low probability that it's right up against the wall because of the observations it's taking. And we're going to revisit this example at the end of the presentation.

Particle filters. So the method that we're going to teach you has four easy steps. One is not repeated. So the initial step is your first sample of particles. If you don't know anything about your state at the start, then you can sample just a distribution of your entire space. And then the repeated steps are updating weights, resampling, and then propagating dynamics.

So we're going to show you a toy example. This is a really simple example just to get the idea across. We're focusing on just one dimension. We have an aircraft at constant altitude. So take that variable out. The only thing we're trying to localize to is a distance x over a map. The sensor is just a range finder down to the ground at this constant altitude. And then the math is a known mapping of x location to ground altitudes. And we're about to see what that means. The goal here is determining where you are in the mountains.

So here we have our plane. As you see, we know the map below, and we know that it has to be flying at this altitude. So with a range finder, you can measure different depths down to the mountain. So here we have long distances, a medium distance, and a short distance if it's directly on top of a mountain. Constant altitude, unknown x. That's what we're solving for. And then we actually have a noisy velocity forward. So we know how fast the plane wants to be traveling in this direction, but obviously with some noise, it could be travelling a little faster or a little slower.

So the first step, sampling. Our initial belief here is that we don't know where it is with respect to these mountains. So we're going to actually sample with a uniform distribution. Well, this is as uniform as I could get it. So over this range, those were our initial particles. Essentially guesses for what the x location is.

We're going to update these weights based on the observation. So we get our first observation from our range finder, and it's this length. So it's a long length. It's, I guess, green in this case. This is our measured value. Here are all the values that we expect to see at these particles. So this is pretty easy to calculate, based on the map that you have. The position of each particle maps directly to a range finder measurement.

So we're going to compare them. So the likelihood of getting these. So measuring the likelihood of each of these occurring. And we're actually weighting the particles. So based on what we measured, these are most likely, so they get a larger weight. And we are most likely not on top of a mountain at this point. So those get smaller weights.

So we're going to resample, given that. So we're going keep the same number of samples the entire time. That's just how this particle filter works. Except we're going to resample across our new distribution. So these are the weights that we just came up with. And we're going to resample over them. So now it's more likely that it's here, or here, or way over there. And then the final step is propagating.

So this whole time that we've been filtering and updating our weights, the way forward. So we need to take that into account. So we have a forward velocity. Let's say this range is 0 meters per second, to 5 meters per second, to 10 meters per second that you can see on this plot.

So we're most likely moving five meters per second. So essentially, this is your change in time between sensor measurements. Obviously if you're only getting sensor measurements at 10 hertz, then you can propagate in between each of those. So here we are.

Using our new sample particles, propagating them forward. For example, this one moving there is a low likelihood. That's a large distance. And these moving at a shorter distance is more likely, given our model. So we have those new ones. The new weights are now based in the probability of it transitioning to that point. How likely is it for the plane to move that far, essentially. And then we repeat. So we're repeating the steps again. We're going to get a measurement in from our range finder. We're going to compare it to the map, to our particles, keep the ones that are most likely and weight them more, and then propagate them.

So as an example, here's time step 2, when we're measuring the uphill of this mountain. Time step 3, now we're halfway up the mountain. So positions that we've kept are at similar heights, so here and here.

And then we can slowly get down to differentiating it. So now that we're on top of a mountain, the only pattern that matches that is here, and maybe over there. And then eventually, we get down to these two.

And then eventually, as this mountain drops off, it's a unique position, where it goes farther than that one did. So in the end, our particle filter thinks that we're right around here or here. And finally, there's a pretty high chance that we're over that valley.

So we'll looking at this, again, now that you know how to do the filter. And this will make a little more sense now. So they started with the uniform distribution. They had no clue where they were at the beginning.

And as the robot moves forward, they are resampling. And the measurements are changing, obviously. Because it's seeing these two walls, and eventually, it sees that there's doorway to the left. And you can keep going forward as well. So eventually, that geometry doesn't line up with any other spot on the map.

Here, we see it nose into this hallway. I think this top hallway is the only one on this map that's that long. But it still don't know which direction it's going. It hasn't been able to differentiate that yet, but it's about to.

So the only remaining particles are here and here. It knows it's in the middle of a hallway. And it knows it's moved about two blocks now without seeing a wall directly in front of it, which that doesn't occur anywhere else, without having another turn-off. So it's about to solve itself.

Because eventually, it sees this wall over here. And those observations don't match up with what it's actually seeing. So there we go. There's a [INAUDIBLE] particle filter successfully working for a little robot with a rangefinder. I went too far. There we go.

DAVID STINGLEY: I'm David Stingley. And now, after we've gotten an idea of why we want to use semantic localization, and how to use particle filters to get an idea of our guesses for initial positions, we're going to talk about how we can use these two to actually implement semantic localization onto a robot. So hearkening back to the implementation idea, we have three important parts.

We have a belief representation. We have the actuation model. So once we have a location, how are we going to move? As we said before, if you don't exactly know how fast you're moving, there's a probability you move to a bunch of different locations. And then finally, we have an observation model, which is the probability that you see some certain observation, given you're in this newly simulated position.

If we continuously solve for our most probable x, then that's our location. So that particle that is the most probable is the place that we guess we're going to be. So let's look at a pseudocode for what a semantic localization would look like.

So as long as our robot is moving, we're going to make observations. And those observations are going to be [INAUDIBLE] z1. Then for those observations, we're going to start off our particle filter and guess a certain number of probable locations. We're going to use our actuation to update it. And then we're going to simulate observations at that location, and compare what we expect to see for that particle on our map versus what we actually saw that we made our observation.

This is going to be our update. And this is what we're going to focus on understanding, since a lot of the rest of this is covered by a particle filter. So what kind of models can we pick, because we need to define what our observation looks like.

And you get a lot of choices. In metric localization, you'd use a labelled laser scan. You'd have perfect information about the environment. And so you can see everything.

It might be nice to use a scene of objects at specific locations, but that requires once again, a lot of information. Now you need to know where the objects are. You need to have an idea of

its exact specific locations and orientations with respect each other.

Maybe it might be nice just use, like, a bag of objects. I see four things in my view pane versus three. These are all choices, and we're going to take a look just really quickly at what the facts of these are.

As we've stated before, there's a lot of choices in observation. It can get complicated really quickly. So just to impress that upon you, imagine if you used laser scanners when you have three objects here-- for instance, a house, a couple of trees, and a mailbox. You check each line for an intersection. And then you have to figure out what counts as your detection, since you're going to have to differentiate what's in your scene.

The problem with that might be is that, well, you saw an entire wall, for instance. Where do you want the house to be? So let's say we assume that objects were a point at some point.

You either completely see it or you completely don't. That way, you don't have to half-see an object. You just check to see if whatever you want your center of mass to be intersects with your current view plane. If it does, then you're good.

The issue with that is that, for instance, something center of view isn't inside the scene anymore, you just completely ignore it. We have the exact opposite problem. So you might want to make it a bit more complex and have polygons, parts of objects.

Do you see some percentage of something? How much of it is in the view plane? It adds in a lot of chance for errors. And that's, I guess, the big point here to remember, is that depending on how we characterize our observations, we have different ways to get things wrong.

So let's say, for instance, for an observation like distance and bearing to some new scenic object, what can be wrong? Well, you have noise inside of your sensors. Your sensors might have limitations.

You can't necessarily see to infinity. So an object might be too far away for you to identify correctly. It might be rotated in a way that you can't necessarily interpret it correctly.

How about if you want to have your observations in classes of objects, so of just everything being an obstruction, now you have different types of obstructions? Trees are different from mailboxes, of course, in which case you have a classification error. What if you see a tree and you just think it's a really, really big mailbox, or you see a mailbox and you think it's a really

small tree with a funny, little, metal top?

These kind of errors can then make your scenes look incorrect. If you decide to have sets of objects, well, what permutations matter? If you don't have a way of differentiating elements in the set, then you don't know if two trees and a mailbox with the trees on the left and the mailbox on the right or vice versa aren't the same thing?

So with that in mind, we're going to be talking about how we're going to solve this question of given some position and your synthetic view, how likely is it that it's your actual observation? And for that, we're just going to change what this probability statement looks like to make it a little more concrete. In this case, Z is the set of observed objects that you have. So we're going to say that there are objects in the scene, and we put them inside of this value, Z.

So for instance, you see a house and you see a mailbox. That would be Z. Your set of objects is just two things. We're going to use the bag of objects approximation.

Y of x is going to be the set of objects you expect to see given your map for the position that you're at. So given a position X, Y is a set of objects that you would see. So you might be in a position where you can see a house and a mailbox. Then Y is is a house and a mailbox. And X is just going to be your position. That's the element that we're getting from our particle filter.

So in our example, as you can tell by how much I talk about it, we're going to use just two things, trees and mailboxes, because I like them. So here's my map. There's going to be a long road, and off to the side, we have trees and mailboxes.

And let's say that your robot wants to be a paper delivery boy. So he needs to be able to figure out where the mailboxes are and how far down the street he is. Because if he were to throw the paper wrong, he'd throw a paper into a tree.

So in this world, our robot is going to be here, represented by an orange hexagon. And it has this field of view. So given this field of view, what does Z look like, our actual observation, one tree and one mailbox?

Well, for this case, what we're going to say is, we're assuming that as long as the thing intersects with our field of view, we're going to see it. So simple finding that. We're just going to say that we see both trees and this mailbox. Once again, we talked about how difficult it is to do this observation. So for a simplifying assumption, let's say that we just completely see the tree.

So that's great for our actual robot position. But when we start spawning particles, we need to figure out what we're going to say they synthetically do. So say we spawn a particle here, and we spawn one when we're just slightly off the road.

We deviated a little bit. We're further forward than the actual position. We drove into somebody's house, another guy's house in the woods.

Then what is each of synthetic observations for these given points? Determining this is going to determine how we actually get that probability calculation. So what do we need to consider here?

Well, the first thing we need to consider is classification. Like we said before, with this set of objects approximation, it's important that you understand if you classify things correctly or not. Past that, like we asked before, we said, wait, why didn't I just see a tree and a mailbox? Well, we need to know if we saw everything inside of our field of view. Maybe we did miss something. Maybe instead that old scene saw just one tree, one mailbox, even though the other tree intersected it.

And noise can happen in reverse, so we could accidentally see a tree when there actually isn't one. And finally, we could see things overlap. We kind of ignored this before, but what if two trees were right on top of each other? It might make it kind of difficult for you to successfully see that there are two trees there instead of one.

So to start with, we're going to strike this assumption. It will become more evident later why this is important. But for now, we're just going to assume that every observation corresponds to only one object being seen. Otherwise, you could end up infinitely expanding your scene.

Think about it. You see a tree. There might be a possibility that that tree is two trees. Well, you saw two trees, then. So maybe there's a probability that each of those two trees is also two trees. And you keep going and keep going, until the entire forest is behind one tree.

It would be kind of bad for doing probability calculation, because you'd eventually have to cut that off at some point, so that your algorithm actually finishes. So for our purposes, we're just going to cut it off at the start and say that everything's just one object. It's just error.

So now we need to talk about if we classify it correctly. If we can solve this equation, what's the probability that our classification is right? So for now, we're going to make two simplifying

assumptions. They're going to remove the two problems that we had before. And don't worry, we'll relax them later.

For our first assumption, we're going to assume that we see everything inside of our field of view. So that means we're not going to have any misconceptions. And we never see something that doesn't exist. So we have no false detections. Everything that's in the scene, we see successfully. If it's not in the scene, we don't see it.

So given that, and we spawn a robot here, and it has this field of view, What does this robot see?

AUDIENCE: Three trees.

DAVID STINGLEY: Yes. It happens to see three trees. So remember our assumptions. We're going to say here that our actual observation for wherever our robot is is one mailbox, and two trees. And we can see that the synthetic robot that we made saw three trees.

So what are some other forms of Y that we can make that would also map to this? What's the way that we can take our Y and transform it so that it maps this Z? What kind of action would we have to take on this?

If you were thinking that we'd have to misclassify one of these trees, you're correct. And remember, this is just a set of objects. So this doesn't have to be the first tree that got misclassified. There's three of them. We could have any permutation of these trees get misclassified.

So it becomes important. And we're going to introduce this concept of this operator, phi. And what phi is going to do is it's going to be a way to map the permutations that we could have of misclassifications for Y to look like Z. That way, we don't have to try and write all the permutations down. It's possible to do this essentially with a matrix, like a permutation matrix, that just reorders the elements that you have.

So what does this probability look like now? We're going to use the lower case z, y and i to represent each individual element of those sets. So for some element in the actual observation, z, what's the probability that some element in our synthetic observation matches it?

Well, for that we need to pick-- well, there's some probability of being wrong or a

misclassifying, or classifying correctly. So we're going to use c to represent a classification matrix. So there's some probability that we classify correctly, usually a higher probability, and then some small probability that an object becomes another type of object.

So how often we misclassify is represented here. But we could add another term. Let's say that our classification engine gave the weighted values of things.

It's common for neural nets and other types of systems that observe images to kind of give weights to their classification. So you might want to use those weights to represent our confidence. And there's a problem that that confidence determines that our classification might just be wrong out the get-go. In that case, we want to know what's the probably that the score is likely to be that type of object.

And then we could have other things. For instance, let's say that our classification was way better at identifying mailboxes from the front. And if we turn the mailbox to the side, it gives poor observations. And it tells us if it thinks that the mailbox is on the side, so it doesn't really know.

In that case, maybe having the bearing of the object inside of our synthetic view allows us to determine another probability for misclassification. The important thing to notice here is that we can keep adding more terms to this. The more specific your classification can get, the more terms you can introduce into it. So add some confusion to it, and make the probabilities of misclassification smaller, and smaller, and smaller.

So with that in mind, we're going to look at what the probability for these entire sets is going to look like. And what is really is going to be is it's just going to be a product over all these classifications for some selection of phi. So you're going to have to take all the different permutations that you could possibly have, and for each of them, you're going to multiply all of these probabilities by each other.

In the end, it's going to give you that entire probability. So it's all the permutations that you have, and then just the probability of each of those objects being classified as that type-- that you just map one set to the other set.

So now let's take a look at another scene. So we spawned a particle above, fake robot here. And it has this field of view. What does this field of view look like?

If you said it looked like two mailboxes and two trees, you're right. And so we're going to assume that we have the same actual observation. We still see a mailbox and two trees.

And we're going to remove our old assumption. We're going to say that it might be possible that we don't see everything in a synthetic robot's field. In which case, how can we amp this synthetic observation to the actual observation?

Well, we just add in a probability that we miss identifying an object. If we just didn't see one of these mailboxes, it looks exactly like this. So for that, we want to capture what's the probability that we see nothing? So say we have a synthetic view with some number of objects. What's the probability we just miss all of them?

So the probability that we miss all of them, we're going add an assumption here, which is that we're going to say that there exists some probability that we see an object for a given synthetic view here, and that we don't see it with a probability one minus that probability. So essentially, there's just two states-- either we see the object or we don't see the object. And we're going to say that probability of identifying objects is independent. So if we see one object, then that does not change our chance of seeing the next object.

Both of these assumptions help simplify the math here. In reality, these might be strongly interlinked. For instance, if your robot's camera is broken, the probability that it doesn't see one object directly correlates with the other ones, because it won't see any of the objects successfully if it has no camera any more. If your robot drives into a wall, and can't see anything because it's staring straight at the wall, the same kind of idea holds.

But for the purposes of making it so that you don't have a lot of covariances, and a really, really big conditional statement of a lot of probabilities of seeing things, we make these assumptions for our items to be independent. Independence means we can multiply our probability successfully. So if we just don't want to see any of the objects, we just take 1 minus the probability of seeing the object for all the objects in the scene.

So what goes hand-in-hand with not seeing anything? Think about if you had a robot far off in the distance over here. What can it see? Just two trees.

So now, we're going to remove the idea that we can't see things that don't exist. So to make two trees map to two trees and a mailbox, we just made up a mailbox of some noise. So what's the probability that we see a full scene when we can't see anything?

And if you're thinking it's going to map to a very similar formula, you're kind of right. But we need to figure out a way to capture our noise statement. Specifically, we are going to use noise as a Poisson variable, which means that there's always some probability of seeing an object out of nothing. And it's going to be coordinated and according to this factor Kz we'll get to on the next slide.

You could choose a lot of different things to represent your noise in this case. A Poisson variable was just chosen by the specific method that we used and implemented from another paper. But with testing, you could find that different potential distributions map to your noise for your particular sensor better. So with a Poisson variable, what we're going to have is we're just going to have the product of all of our Poisson variables times this Kz factor for the given scene that we want to map to. Essentially, it's just the product of all these independent Poisson variables for each of our different objects that we want to create.

So with that in mind, what's this Kz factor that we're multiplying everything by? What it's actually going to be is it's going to be a set of uniform distributions. These uniform distributions are going to be uniform over all the classifications we could get for an object we spawned from the noise, and all the possible scores, and all these possible bearings for this synthetic object.

And if you remember a few slides ago or you rewind the video, you'll notice that these map directly to the categories that we put into that classification engine. In fact, they should. So if you added more things or took them out, you changed this uniform distribution.

The idea is that when you synthetically create an object out of noise, you might get any of those types of objects. But if you needed to create a tree synthetically from seeing it in the noise, you might get a tree or you might get a mailbox. Either one of them might show up.

If you had a different or more intelligent distribution for how you might misclassify things-- for instance, let's say your noise, whenever it shows up, always identifies trees. Whenever you accidentally see noise as an object, it's always a tree. Then you might only want to have a probability of seeing trees over here, and it would change you distribution. But for simplicity's sake, we're going to assume that it's equally probable that any type of object shows up out of noise.

So now, we're going to put it all together, since we relaxed our assumptions. So we have a lot of different things that can potentially map to this actual scene. We have scenes that lack

objects.

We have scenes that lack objects and therefore, need to add the object in. And when they lack objects and have to add the object in, there's a chance they add in a different object. And then maybe, they just misclassify something, like our misclassification idea.

But wait-- I guess we can make this more complicated. What if one of them just wasn't seen, so we'd take one of these out? And then we just see two things from noise.

As you could see, these keep getting more complicated. But if you think about it, with our previous probabilities, is every single one of these is going to add a lot of probability terms to be multiplied by each other. You can keep getting as complicated as you want it to.

If we removed our first assumption, we'd be adding in probabilities of two objects becoming one object. The idea is like higher power terms when you're doing approximations. The goal is to make sure that any particle you spawn could potentially be what you've actually seen. But we want the ones that are very low probability to really be very low probability. So we make sure that these incredibly complicated transformations are going to be so insignificant that they'll usually return to almost 0.

So if we wanted to solve this, we need to start by using a little assumption, which is the fact that we now have to fold in the idea that we could have some missed detection and some false detections. So false detections would increase the number of objects we've seen over the number of objects that are actually presence. And missed detections would reduce the number of objects that are actually present.

Notice that this always has to be same size as the number of expected objects, because you're trying to map whatever your synthetic scene is to the actual thing that you observe. When you do this, it really turns into a large number of multiplications. So this should look familiar. This is our term for successfully classifying all of our objects, and we're going to multiply it by the probability that we actually identify it, since we could now miss things.

Then, we have to multiply that by the probability that for whatever objects we didn't see, that we actually missed them. And then finally, we have to add in the noise, because the noise is going to make the objects that we're missing show up, so that we have a classification of the same size as the thing that we want to see.

Now, one thing you should note is that we added in phi way down here at Kz. That's because

we have to map these false detections that added to the number of objects that we could see, as well as the actual detections. Essentially, we have to take all the objects that are seen here, real or not, and map them to all the objects we expect to see, because they're going to be one to one.

So let's take a look at a little video that shows what semantic localization can do. So in this video, for a little heads up, we have essentially a scene where they have mapped out two things inside of a suburban area. They've mapped out cars and windows on the path of the robot as it drives around an area. And they're attempting to use just the information of the cars and windows visible on the scene in order to localize where they believe the robot to be during its journey.

[VIDEO PLAYBACK]

- This is a video extension to the paper--

DAVID STINGLEY: So going to mute that. It's on. So this is a few steps into the process. And then it resets. It's spawn a number of points for potential locations. And then it very quickly localizes and finds its spectacular rotation.

These are all the identifications that are happening inside of the scene. You can see the boundary boxes for cars and windows showing up, the cars in red, and the windows in green. It expands out its distribution of particles.

And then shortly thereafter, it gets a couple of approximations. And then it settles in on a location. It happens to use a kind of like a centralized weight for where it is for the set of particles, and then it draws the car as being in that location.

If you let it keep running, it occasionally expands back out to make sure it doesn't fall into a local minima, and then compresses once again to where its strongest belief is. Notice that it has a couple of seconds of having a very, very spread distribution, and very quickly converges into a singular point. That's because a lot of these locations become very, very low probability after you've seen a couple of scenes into the future. And we're going to pause this.

[END PLAYBACK]

I welcome you to go see the video, to see the video yourself if you want to. You can check the title. It's pretty easy to find on YouTube, because it's pretty much the only one. So hopefully, it

works now. Step back over here to the other side.

So why would semantic localization be useful in this manner? In the example that was shown, it was kind of done on post-processed data. They took a scene. They drove around it already. Then they did the identification on the video feed, and used that to do a localization.

We talked about before. But people kind of use sparse information to do a lot of their-- or if you can't walk into a room-- people don't walk into a room and produce an exact laser scanned map of the entire area themselves. But they can store important information, like seeing objects and tables that they find with their shins, and use those to move around.

Robots store that perfect map, if they can manage to make it. But they don't really have a good understanding of what that map might mean to a human. So that means that we're actually kind of like better at doing tasks with the environment.

If we wanted to go directly to a location, we don't have to look around and figure out on a wall what our distance from the wall is before we can then localize and move ourselves back over to the table. We just say, oh, I want to go to the table. And you look over there. Oh, there's a table. Table, got it.

So how can we make robots think a little more like humans, so it's easier for us to give them instructions? If we can make the robot use a map that has just scenic objects like we use a map that just has scenic objects, then order to move between scenic objects are as simple as turning, finding the object, and saying, oh, well, I know where the object is. So I know roughly where I need to be. And then we start moving in process towards it.

In conclusion, semantic localization has a lot of-- I'm going to leave the references slide up while I talk, so that if you wanted to go and take a chance to see any of these papers, you definitely can. Most of the work in this slide comes directly from these sources. Semantic localization offers us the opportunity to take robots, use sparser information to potentially localize, find ourselves inside of spaces.

It also gives us a chance to have really tweakable factors for how you might understand where you are in a space intuitively. If you think certain things are important, you can add them in as probabilistic factors. If you don't, you remove them just as well.

Thank you. And this is essentially the conclusion of our presentation. I definitely recommend

taking, if you wanted to use something like this, taking a look at this particular paper here. Notice how it says, "via the matrix permanent?" As I said before, a lot of these operations are a series of multiplications and some permutations.

There's permutation matrices. There's matrix multiplication. And there's a lot of ways to make matrix multiplication faster.

This paper details how you can take the math that was shown before that's pretty inefficient, and turn it into something that you can do an estimate of very quickly.

[APPLAUSE]