PROFESSOR 1: So as he gives an introduction, myself along with Catherine, David, [INAUDIBLE], and Charlotte [? are going to present ?] you guys Probablistic and Infinite Horizon Planning. To give you a brief overview of what we're going to talk about, we're going to start with the Quadrotor motivating example. We're going to move into planning with Markov decision processses, give you a little bit about value iteration before discussing heuristic guided solvers. And we're going to go into the more stochastic case, partially observable Markov decision processes and operating in belief space.

So we very often now see quadrotor motion planning as a problem, given, for example, with the Amazon fulfillment center. We start with a goal configuration, start configuration, set of actions we can take, and some type of reward function or cost function.

So for instance, if we have a quadrotor starting at the Amazon fulfillment center, and we want to get to 77 Mass Ave, and let's say we want to take the shortest path to get there. We would follow the red dashed line. But, as we can see, it comes very close to these obstacles. So we're looking at very higher risk for mission failure, crashes. If there's any uncertainty in its path, we're going to have a problem. So one of the ways that we can compensate for that is we can also done a green path, which is adjusted to give us a little bit of space. Are there any more questions?

So as you can see, the level of uncertainty allows us to determine how easy or difficult the problem's going to be to solve. On the easier side we have deterministic dynamics and deterministic sensors. In this case our actions are going to be executed as commanded, and our sensors are going to tell us exactly where we are. This will be something like dead reckoning validated through sensing, very redundant. If we moved to a little bit more at a difficult case, we have deterministic dynamics. Our commands are still being executed, but maybe we have a noisy camera. So we have some uncertainty in the sensors. These are the cases where we would see dead reckoning, but we would compensate with Kalman filtering to get rid of that noise level.

Now, down at the-- yes?

AUDIENCE: Sorry, what is dead reckoning?

PROFESSOR 1: It's essentially-- you are saying I want to execute this option, and it's going to execute it exactly.

AUDIENCE: OK.

PROFESSOR 1: Then towards the bottom, we have stochastic dynamics and deterministic sensors. So in this case maybe there's some uncertainty in our actions. But we can validate through sensing. This is where we're going to spend the next section on Markov decision processes. And then briefly, later in the lecture, we're going to talk about this most difficult case, which is the stochastic dynamics and stochastic sensors. Our execution maybe, maybe not, our sensors maybe a little bit of noise. And this is where we see partially observable Markov decision processes.

PROFESSOR 2: Can we make just a brief clarification of the dead reckoning? So dead reckoning is where you estimate you position using a probabilistic file, but you don't use any observation stuffs. That's the thing.

PROFESSOR 1: OK. So we talked a little bit about action uncertainty, where we're going to focus. And this is a case where, for instance, even if you tell a quadrotor to stay in its place and a gust of wind comes by, it's not going to stay in that same place, right? It's going to have a little bit of movement. And we can see, this causes things in disparity where you have a command of trajectory and your actual trajectory and then you need to associate. So in order to compensate for that, we want to model things with that uncertainty or else we have these higher situations where we command to follow some line. We don't incorporate the uncertainty, and we see a crash.

So this allows us to introduce Planning with Markov Decision Processes. So MDPs have a set of states-- some actions you can take-- a transition model. So essentially, what's the probability of reaching some state if you take an action? An immediate reward function and a discount factor. This discount factor is important because it allows us to prioritize gaining an immediate reward as opposed to an uncertain future award. So the concept of, bird in the hand is worth two in the bush.

Now we want to find an optimum policy that will essentially back an action-- the best action-- for each state. And what we hope to get from this is maximized expected lifetime reward. So we want to maximize the accumulative reward we get over the times. So let's walk through an example. If we have a quadrotor with a perfect sensor, and let's put it in this environment [INAUDIBLE] 7x7 bridge. Our set of states are obviously [INAUDIBLE] space in them. Can anybody tell me what some of the actions might be?

AUDIENCE: [INAUDIBLE]

PROFESSOR 1: (LAUGHING) Yep. Up, down, left, right. In this case, we call them North, South, East, West, or Null. The next thing we need for this example-- arbitrarily gave ourselves a transition probability. So we said that you have 2% chance of following your commanded action with a 25% chance of moving to the left or the right.

Next we have the reward function. And again, we arbitrarily decided that we want it to be a reward. If you get to the state 6-5, the increment [INAUDIBLE]

AUDIENCE: Alicia, you had [INAUDIBLE] left or the right, is that clockwise or counter clockwise? Let's say you had planned to go to the right, would that mean you have a 75% [INAUDIBLE] or 25% chance of [INAUDIBLE]? What if you just waited [INAUDIBLE]?

PROFESSOR 1: We said clockwise, counterclockwise from the intended direction of action.

AUDIENCE: Thanks.

PROFESSOR 1: Uh-huh. And finally, we give ourselves a discount factor of 0.9. So let's assume for a second that we have our optimal policy. And let's say that our optimal policy says, from this state, we want to take the action North. Right? As we discussed, we have a 50% chance of going North and 25% chance of going to the left or right.

So after that time step, these are possible states that we could end up in. Right? So now let's assume for a second that we can take our next action. And our next action says, go North. Again, we have the same probability distribution. And these are the states we could end up in after two time steps. We can see that this starts getting very complicated, right? And there are increasing amounts of uncertainty. So does anybody have any ideas on how we could collapse this distribution? Keeping in mind that our senses, at this point, are deterministic. Yep.

AUDIENCE: Fly to a corner?

PROFESSOR 1: I'm sorry.

AUDIENCE: Fly to a corner?

PROFESSOR 1: We could do that.

AUDIENCE: You just sense how far away the red box is.

PROFESSOR 1: We could do that.

AUDIENCE: Quick comments. So the blue state is not actually one.

PROFESSOR 1: I'm sorry.

AUDIENCE: So the problem is you're not actually one.

AUDIENCE: Yeah. The issue is if you went to (1,3) and then you transitioned to the left, that's where the 0.0625 is coming from.

AUDIENCE: [INAUDIBLE] then, shouldn't those numbers always are actually 1-- your probability distribution always are actually 1?

PROFESSOR 1: Yep.

AUDIENCE: So it's just a point. It's just off the screen.

PROFESSOR 1: We would add another section to the screen and just move the grid over. We just cut it off for graphics.

AUDIENCE: OK.

PROFESSOR 1: Yeah. So those are all great points. The easiest way to do it is just take an observation. So at this point we say, after our first time set, we weren't sure which of these three states we were in. So we took an observation and said, wait, we're actually here with complete certainty.

So to make this a little bit clear, we're going to look at it from a tree view. Right? We said that we started at a state. We took an action. And this is our possible states we could have ended up in. We're going to collapse this by taking this observation. And now have a complete study here. And we take our next action and see that we have moved out here. So this allows us to basically ignore the history of states. We have the same percentage probability from each time

set. This will be really useful in completely collapsing the distribution every single time you take an observation. Anybody have any questions on that? OK.

So let's go back now and figure out how he came up with his optimal policy. They way we do that is through dynamic programming. There's two different ways. You can do it either through value iteration or policy iteration. For this lecture, we're going to focus on value iteration.

So let's take this same example we had. We still want to maximize the expected reward. And so to start, we're going to initialize the values of each state to 0. Let's for example, start at (2,0). We're going to focus on suite 6-5. And we're going to say that we're going to take the Null action to start with. From there you can see with a probability distribution 4, 6, 5, we have a 50% of staying. We have 25% chance of going to 5-5. And a 25% chance of going to 7-5.

The next item of information is the values at each of those states that we could end up in. And currently, they're all set to 0. And finally, the most important part here will be with the reward [INAUDIBLE] 6-5, taking the null action, regardless of what state you end up, is going to be 1, as we defined in our initial set up.

So let's see how we would calculate the next time step value of the state. You'd start by taking probabilities, right? And from there, we would add the reward here. And because we said that the reward does not dependent on the state we end up in, [INAUDIBLE] should be across all three probabilities.

From there, we're going to add in the discounted value that we had from before. So a way to look at this in a more generalized form is to say that across all the states that you can end up in, you're going to look at the probability of ending up in that state. You're going to multiply that by the sum of the reward and the discounted lifetime value that it has.

So we want to make sure that we're getting the best possible values. So we need to incorporate all the other actions that we can take from that state. So what's going to happen is we're going to take that general formula, we're going to repeat it over all of the possible actions. And then we're going to take the maximum of that. So, for this example, the state is very easy. All of them are the same for this case. But we go fast, and we say we get a value of 1. And we update it by showing the graph.

This gives us what's called the value [INAUDIBLE] Backup-- or Update equation. This will be really important because it reaches across the entire states space and allows us to provide a

history.

So what this would end up looking like is we're going to iteratively calculate the values across the entire state space So at t0, we determine that all the values are 0. At t1, (6,5) gets a value of 1, and at t2, we see that value start to propagate out. Make sense so far?

So the way this works is you would repeat those iterations until your changes in value become what we would consider small enough, which would indicate your approximation is close enough to the real value. From there, you would extract the optimal policy from the lifetime values. So you see the [INAUDIBLE] in the Bellman equation. And you're now just taking the action from it. And you would map those actions to your states.

An example of what this might look like propagating out-- if you say blue was the reward and red is an obstacle, et cetera-- you can see, as that value propagates out, you start seeing your policy by the arrows. Anybody have any questions?

So one last thing to mention about this, though, is the complexity for each iteration is dependent on the size of the state space squared and the number of actions you can take. So you can imagine, as your state space expands or you gather more actions it's very complex, in which case time and value depiction becomes very time intensive and costly. So this allows us to move into the Heuristic-Guided solvers.

AUDIENCE: Thank you. [INAUDIBLE] transition, one quick question. Can I just get a show of hands-- how many of you have learned value iteration before versus how many of you have seen it for the first time? So how many have seen it before? OK. And then, how many is this their first time? [INAUDIBLE]

PROFESSOR 3: Any questions on value iteration before we jump in? It's going to be an essential part of how we're going to do [INAUDIBLE] solvers. Anyone who hasn't [INAUDIBLE]. So the most important thing we said about value iteration is that it's super slow. It's going to have to go over every possible state and every possible action that it can take. Our state space is multi-dimensional, and we can take a lot of different actions. That going to be really costly and really hard to [INAUDIBLE]. So the approach we're probably going to want to take is using some sort of best first search applet? Who can tell me some example algorithms that already do that?

AUDIENCE: A star.

PROFESSOR 3: A star. So that's very good. And that's exactly what we're going to base our stuff off of. The A

star is for deterministic graph search. If we have a graph, we can use it heuristic, and we walk down it and search the space that we're most interested in until [INAUDIBLE] variable. So going to introduce to new items focusing on the last one. AO star is an algorithm we can use to search for graphs that have this "and" problem. AO stands for And Or graphs as opposed to the simple graphs that we have [INAUDIBLE]. The And is a way to express probabilistic coupling between edges. So if we explore one thing, we might have to explore other functions. We'll discuss that a little bit more in the next slide.

LAO is a bit more of a generalization. What it does is allows us to search loopy graphs and deal with the probabilistic coupling. And allows us to search and find the best path with a heuristic guided algorithm over infinite time horizon, possibly revisiting states using the tools we just had-- valued iterations-- to understand what the next best thing to do is. So we'll talk about exactly how to get our MDPs that we just saw, these little arrow examples to these And/Or graphs.

We'll talk about two cases. One simple one where we could apply the AO star algorithm is where you have a qualicopter. And if you command an action North, you have a high probability of going North, but you might go East. And vise versa. If you go East, you might go North. This is expressed here with the growing tree. And these And edgers that connect it. But despite the action of reading my command from 0 to 0. We might end up in (1,0) or (0,1). Right? As we propagate outward, you can see how we're never going to loop back to a statement we've been to. We're going to be moving in the Northeast direction constantly. Our [INAUDIBLE] and our probability distribution across this tree explands.

And that's the coupling of the edges. Because as we explore down the tree, we have to explore all the edges together. Anyone have any questions on just this kind of conversion formulation?

AUDIENCE: I have a broader question. We mentioned that MDPs deal with finite states. Do we always just discretize a continuous row into a set of planning states?

PROFESSOR 3: Yes. That's our prerequisite for searching over the state space. We can do that as finely as possible, but yes, discretization is there.

So now let's look at this case. Instead of the Northeast, let's say it's not deterministic whether we command an action north or South that we go north or south. Here we see this loopy

structure begin to emerge. We see that we might-- on our first action commanding North, we might go to plus 1. And then commanding North again, but we have our And edge and our probability distribution is [? back ?] to 0. What this does is this loopy structure. And this is exactly what we're going to be exploring. This is a very real world scenario where it's a very likely we might return to somewhere we just came from, just because of the uncertain dynamics we have.

This is the type of problem [INAUDIBLE]. So we're going to use the LAO to start out with. We're going to talk about the three main things that [INAUDIBLE] has a heuristic guided envelope. And what that means is that we have our large state space here. But we're only going to look at a portion of it. This greyed-out portion. We're only going to look at the portion that is interesting to us-- the portion that provides us with the biggest rewards-- the portion that's reachable if we follow an optimal policy.

We figure this out with some admissible heuristic. We'll estimate our rewards just like A star. The idea here was that we'll keep the problem small so we don't have to search the valuation for a giant state space. What we'll do next is at the state space expands, we get a bigger picture of the states that we're interested in. We're going to do [? an audio ?] [INAUDIBLE]. And then we're going to figure out what the best action is. And in the ideal case, the states that we would never reach using an optimal policy are never going to be explored. Because our policy is going to say, no don't go over there. That's a dead end. Or that's getting you farther away from [INAUDIBLE]. So we're going to be searching in a very specific part of the state space that is useful to explore-- that will get us closer and give us a higher reward.

What's important here, the L stands for Loops. It's an extension of the AO star algorithm, which is by itself is an extension of the A star algorithm. We can handle infinite horizon problems, like transition in different ways. And really models the real world scenarios we're interesed in. Any questions so far on the broad scope of what AO star is going to do? Yeah.

AUDIENCE: Can you put the [INAUDIBLE] what you're doing over here, but--

PROFESSOR 3: Sure. The AO stands for And Or graphs. So that's graphs that we have here edges that are coupled. If you took time to do this transition, you might end up here or you might end up there. So that's the notion that of this probability coupling, that we'll see in action as we [INAUDIBLE] we're going to do, we're going to input this MDP or AO graph with transition probabilities or reward function heuristics. These are all things that we defined prior to figuring

out a plan. What we're going to come out with is an optimal policy for every regional state. It's a little different than what value iteration is, which is an optimal policy from every possible state.

But we argue that that's all we need. If we know where we're starting and we follow our optimal policy, we're only going to explore a certain portion of the state space. And we're going to explore that together. [INAUDIBLE] plan a little bit more efficiently than iterating for a high [INAUDIBLE] heuristic. Any questions on that?

So we'll define some terms that we're going to use throughout this. We've already talked about our state space. This is just a small portion of it that we're going to work with as we're going to walk through this example. Next we're going to define something called our envelope. And that's the sub-portion of our states that we're going to be looking at.

We're going to initialize that to just this zero [INAUDIBLE]. But as we progress through the algorithm, it's going to grow. It's going to grow only in the areas that we're interested in. A subset of this envelope is the terminal states. Now these aren't goal states, these are just [INAUDIBLE] of our space that we've added to our envelopes. And this is what it would be in the expanded [INAUDIBLE]. And it's just the nodes that haven't been expanded yet. Here they haven't drawn anything incredible but you can imagine the state space goes out further because [INAUDIBLE]. You can imagine that this goes out further. And they're keeping track of the nodes that we haven't expanded yet. Or likewise, we've showed that we initialize [INAUDIBLE].

The other few things that we're going to define-- we've already defined the states that are in our envelope. That's the blue or the red. We're going to define cost heuristic or reward function, R E, and a transition probability matrix, or set of matrices, that we're going to use our optimal policy search on. What's important here is that our reward function and transition probabilities are slightly altered to account for the fact that we haven't explored the entire state space. We see here that if a node is in ST, in other words, it's one of those terminal nodes. We're going to say we can't transition out of it, because we don't know what's beyond it so far.

And we're going to set it for reward to be a heuristic. Whenever we think that the reward is going to be when we begin to explore and go further. Like I said, we're just going to feed this into a policy search just like we discussed with value iterations on the sub problem. And we're going to search for an optimal policy [INAUDIBLE]. So far so good?

This is the general steps. This is very text-y, but we're going to definitely walk through every single step. We're going to do two full iterations of the algorithm. So like I said, we're going to create RE and TE. That's are reward function transition probabilities using the definitions we showed. We're going to use value iteration to find the optimal policy of the sub space that we're interested in now. And then this is probably the most important step. Knowing this optimal policy, we take a look at what new states that aren't in our terminal states-- nodes that we haven't explored yet-- what new states we might visit now.

So let's say we have our policy, and it says we'll go North. And we know that we haven't explored the North state yet. We know this is the state that we're going to reach following what we consider now to be already an optimal [INAUDIBLE]. So that's the states we're going to expand next. We're going to do some bookkeeping, adding them to the terminal states [INAUDIBLE] once we expand it, then adding them to our envelope. What's important here is that we're only going to add states that we visited yet to our envelope.

And this is basically the little hack that allows us to deal with loopy graphs. We're not going to continually explore nodes that we might reach a second time, probabilistically. We're going to let value iteration handle that [INAUDIBLE]

AUDIENCE: [INAUDIBLE] states are expanded. I'm the one who got confused on that. Are you saying that you're going to just repeat until there aren't any more terminals to look at. And if that's the case, how is that possible if you have an infinite horizon [INAUDIBLE]

PROFESSOR 3: Sure. So if you can imagine-- and we'll talk about termination at the end. But you can imagine that as we have these terminal states, but you have a policy that guides you to a part of the state space that we've already expanded. Imagine you've reached your goal. Your optimal policy is going to say, stay put. Right? And it's not going to say, move North again.

AUDIENCE: It's just the goal [INAUDIBLE]

PROFESSOR 3: Essentially, the goal state is definitely an example in the more extreme case that there's nothing else you can do that's going to get you closer to our goal. The idea is that your policy on your sub space never tells you to go to a terminal. Nobody can [INAUDIBLE] inherently worse than [INAUDIBLE]

AUDIENCE: Planning optimal policy means running value iteration entirely.

PROFESSOR 3: Yes. We were going to treat it essentially as a black box. But the trick here is that we're doing it on a smaller portion of space of a different world. All right. I'm going to put the steps up there. Hopefully you can see this. But for now, we'll just walk through from the beginning of what we're going to do. So we said that our envelope and our terminal nodes are just at 0 to start.

So very simply, we use these definitions and say, OK, the transition probability as 0 to any node right now is 0. Because it's in that terminal. [INAUDIBLE] That's just so that we don't transition out of it. We can develop the policy based on only this portion of the space [INAUDIBLE]. And our reward function, we're just going to apply it to be the heuristic [INAUDIBLE]. Let's say that's 20. And for a move-on from there.

[INAUDIBLE] started to value iteration. And we'll run this value iteration. So this is a very basic case. We're just going to-- we're using this to kind of build up the machinery, understand what we're doing. It's a very basic case where you only node we have is that zero. We can't transition out of it. All we have is some heuristic. So the only thing we can do is nothing. So the action we're going to take from S0 is nothing. Very simple case just to get us warmed up and understand what's happening.

So using this policy and knowing that those 0s in our terminal node are node reset. We're going to take the intersection between this terminal mode set. And the nodes that we might reach following our policy. And we know that we're at S0. And we know that action that our optimal policy says to take is not. So we know we're there. And we know it's in our terminal node set. So that's the only thing that we could reach so far using our optimal policy. So far so good? Clear?

So that's exactly what we're going to expand. Just expand S0, A, B, and C defined those to our terminal nodes. So that's up there the symbols [INAUDIBLE] from our terminal nodes added as children. Then we added the children to the end. Here we go. We're going to do a little bit more. We're going to do the same thing again. But now we obviously have more nodes and a bigger sub space to explore.

So using these definitions we see our reward function is a tuple of the node that we start from. And so this S0. And one of these three actions. And we'll take A1, A2, and A3. When we have our instantaneous reward functions, 6,4, and 8 as being our rewards from doing those actions. From A, B, and C, no matter what action you take, it's just heuristic. That's part of it. And likewise, we're going to take a look at this transition probability [INAUDIBLE]. This is for

[INAUDIBLE] transitioning from a state S0 for saying that if we take actions A1, A2, and A3, what's the probability of being done in nodes A, B, and C?

If you look so far here, we're going to look at something deterministic. If we take an action, we'll end up where we say we're going to end up. And we'll see how this algorithm collapses down to A star if everything's deterministic. We're also obviously going to look at the probabilistic case where we say [INAUDIBLE] small probability that it might end up with [INAUDIBLE]. That's going to necessitate that we're going to have to look at and expand B together with A if we were to decide we want to try [INAUDIBLE]. And likewise, if we try to take action A3, we have to expand them all three nodes. So the tighter the probabilistic coupling, the more of the space we're going to have to explore.

So just off this, assuming that if you take action A1, A2, and A3, can someone tell me what the policy is for the rewards here? And we're interested in a policy from S0 what we're going to do. Take a look at the rewards and judge what the best action to take is from a purely deterministic sense.

AUDIENCE: [INAUDIBLE] do you add [INAUDIBLE] or do you just [INAUDIBLE]? You have the reward that you have gotten so far.

PROFESSOR 3: Does that help you answer the question?

AUDIENCE: Oh, yeah. [INAUDIBLE] student.

PROFESSOR 3: So the policy preference says, from S0, take action A1. And that's going to stay there. [INAUDIBLE] from A to C, there's no action to take [INAUDIBLE]. What that means is that the nodes that we might reach that are in our terminal state set, using our policy, is node A. All right. So that's the node going to expand. And this is where you really see that all we've done is collapse down to A star. A star would say, OK. What's the best node using some heuristic. Action a1 one takes us to that best node, and we're going to expand just that. So when everything is deterministic, basically this algorithm collapses down to A star. Nothing super interesting. The interesting case does come up when we start doing more probablistic ones. That's where nodes are probabilistically helpful to the scanned graph sense.

So now we have our policy. The policies are politely going to remain the same, because they have very little probability on the edge actions that we might accidentally hit up. What would I want to look at is what [INAUDIBLE] and how reachable following our optimal policy.

So we talked about that if we were to actually read. We know some probability of ending up in [INAUDIBLE] nodes. So taking action A3 makes C, 2, and A all reachable. What's reachable in taking action A1?

AUDIENCE: A and B.

PROFESSOR 3: [INAUDIBLE] And that's where we get the notion on this probabilistic algorithm that we're going expand things together. Explore the part of the state space that we reach both higher optimal polity and that we might accidentally end up in if we [INAUDIBLE] the policy. And this is what guarantees that we'll have an action to take from any state that we intended to go to or that we might end up [INAUDIBLE].

That's how our state expands, our envelope expands to encompass only the reachable and interesting states that we want to look at. It's not as simple as just examining not only the heuristic, but with A start because we have this probabilistic [INAUDIBLE]. You can see that if you have a tighter coupling, then you don't get to exploit this optimization as much. For example, A3 we would have had to expand more as opposed to A1. We can just stick to A and B and ignore expanding C for now. [INAUDIBLE].

AUDIENCE: Our [INAUDIBLE] We know that when people do the statistic [INAUDIBLE]. So in this case, if you take action a because of a and b, [INAUDIBLE]

PROFESSOR 3: So in the complete sense of running this algorithm, we shouldn't print it. Because what if we do? If we have that 2% chance, [INAUDIBLE]. We need to have a policy for correcting. So we end up at B. You can imagine that these are going to try to push back to whatever path that A was looking for. I suppose that the probability is low enough, you can have some cutoff percentage where you've decoupled [INAUDIBLE]--

PROFESSOR 2: So just a quick point. So I think that's an excellent point and an excellent answer. We're going to talk about next week-- exactly what's going to happen and if you can prove lower probability on the paper right here will be lecturing on that. Good question.

PROFESSOR 3: That also gets us into the sense that if every state in the world were probabilistically coupled-- let's say we had some transporter to go with the *Star Trek* examples. If we had this transported that non-deterministically put us in any state in the world, we have to explore the whole world, because we could end up [INAUDIBLE]. So luckily that's not the case yet and we can take advantage of the fact that we ended up most likely where we commanded with some

probability of [INAUDIBLE]

This is exactly what we just talked about. We coupled these nodes with this And edge. And we expand those, too. Is everyone understanding the holding intuition, and the logic for why both of them have the [INAUDIBLE]? Even if there's only a small probability that we end up [INAUDIBLE]?

So and this is what we're going to repeat until the states are expanded. You can imagine that the next time we run our value iteration, we're now running it on all of these colored nodes-- both the blue and the red. We can imagine that next time, now that we have a little bit more information what lies beyond A and B, that our policy might say, oh, you know what? Actually from S0 C, action A3 was the best to take.

What that does is say, OK, with a regional map, it's A, B, and C. And we expanded those nodes. We'll we've already expanded A and B, so we move into the next part of the sub space, that as we gain more information, we run value iteration and we can expand on why.

Steve asked a good question. When we did our dry run, about whether there is a way to save on the computation you did prior to the value iteration and add these [INAUDIBLE] states. And we've looked at it a little bit. I've seen some stuff. But I haven't found a paper that specifically deals with it. You can imagine how you've already run value iteration on your previous iteration [INAUDIBLE] and you add the new terminal edges which you'd expand on. And you run them again until it stabilizes. And that way you've saved the computation of having to run valuation multiple times [INAUDIBLE] state space.

AUDIENCE: So I'm trying to think of how this is different from something like [INAUDIBLE] for [INAUDIBLE].

PROFESSOR 3: I think I don't know enough about that. But my basic understanding says that what's useful here is it's this explicit [INAUDIBLE]. I don't know how much [INAUDIBLE].

AUDIENCE: And also, as long as your heuristic is admissible, it's guaranteed [INAUDIBLE] not all [INAUDIBLE] algorithms. Just like A star is optimal, as long as you've got a consistent [INAUDIBLE].

PROFESSOR 3: All right. So that's definitely the idea here. We coupled these in the only explored of the portion of the state space that we'll reach an optimal policy [INAUDIBLE]. So we'll talk about quickly another determination. We've touched on most of this. So it's most likely when there are no

more states to expand this when we've reached our goal. It's when our policy that we run on our entire envelope from value iteration doesn't say that we should go to anymore terminal [INAUDIBLE] that we haven't looked at yet, that we haven't seen yet.

We've said that those are only the things that are reachable and needed. Both, reachable because we're following optimal policy, and needed, only if we might accidentally end up in the probabilistic. This is gives us the sense of rigorous that we get policy on the entire state space that we can end up in following the optimal policy.

The third bullet here touches on if we don't expand the states that are probabilistically coupled and we do accidentally end up there, we risk getting lost and not having a policy. We can compute this all off line and have a plan before we even start planning to know exactly where we want to go even if our dynamics aren't [INAUDIBLE].

We're come back to this. This was our motivating example. And so we show that these real platforms can be modeled stochastically and then we can pretty easily deal with that. Search our state space and deal with those probabilities and expand the nodes that we might end up. Right? And the heuristic allows us to not have to explore these areas of state space. I never actually end up there. We'll always be a commanding toward 77.

We're never going to try to command backward. Sure, if there's a gust of wind and we have some probability there. But you can imagine that we're going to only explore a small portion of this. Because we'll always be trying to correct to get back to the top four blocks. And using our reward function, we get to determine if we want to fly a quick path or if we want to fly a safer path. For example, our time times our probability [INAUDIBLE] we want to perhaps reduce that.

All right. Are there any questions about planning with MDPs, anything like that. I love this stuff. So the more questions, the more I get to talk. Fine. What I'm going to be talking about for the rest of this lecture is extending beyond MDPs to a broader class of problems called POMDPs, Partially Observable Markov Decision Processes. I love this stuff. I think it's really cool. They're really fun problems. We're going to talk about why they're so much harder to plan with, to execute-- but why they're important to at least know about so that you can model real world problems with them. And then we're going to delve into a case study of a specific POMDP solver. We're not going to go into as much detail as we did for MDPs, but we're going to look at what powerful results we can get by planning with POMDPs.

PROFESSOR 4: So first, I want to rephrase this in the overall talk. Right? We have this spectrum of uncertainty. And coupled with uncertainty is difficulty of planning, of solving, of executing a problem. And we've killed these first two cases. That was really easy. And then we just discussed the bottom [INAUDIBLE], MDPs.

What I'm going to talk about is the case where both your dynamics and your sensors are stochastic. Why is that important? It's because when we first saw this slide-- our motivating example slide, we only saw the left hand side. We said, our actions are uncertain. But good news, we have a perfect sensor-- a perfect camera. But that's unrealistic. I think, we have all, to some extent, experienced the fact that no sensor is totally perfect. Your camera might have fluctuated pixel values. Your laser range finder is going to never read out exactly the right number all the time.

You can have a camera in different lighting conditions that will behave differently. You might not be able to observe your full state. That's, in a way, an imperfect sensor, right? If I'm in this room, I have imperfect eyes. I can't map out all of MIT's campus because I'm blocked by walls. How can you deal with the fact that you can't see all your obstacles all the time? We've already talked about some cases-- that there are some algorithms that can help us with that, like D Star Lite. But can you reason about these things probabilistically? And then finally, you might be in a non-unique environment where you cannot resolve your state with certainty no matter how good your sensors are.

Imagine you're in a building with two identical hallways. You're dropped off in one of them. How can you figure out where you are? You can't unless you start exploring. And so we've got to deal with this uncertainty, right? it's Part of every single problem. When observational uncertainty is slowing, you can maybe ignore it. But it's there.

And so we're going to formulate this as a POMDP, a partially observable Markov Decision Process. And this next slide is just like the MDP slide. Hairy, but important. Right? We can formulate a POMDP, which is seven elements. And MDP too five. Most of them for all those are carried over here. We've got our set of states where we can be. We've got a set of actions what we can do. We've got our transition model which says, given that I started in one state and then I took an action, what's the probability I end up somewhere else?

And like David was talking about, hopefully that distribution is pretty local-- we're not teleporting all over the world. We've got our reward function. This is exactly the same as for

MDPs. And we've got our discount factor down here. The key difference of POMDP is these two elements. We've got a set of possible observations and a probabilistic model for the probability of making an observation given your state and the action you just took.

Now it's important, I think, it matches up really well with real world sensors having this probabilistic model. If you have a laser range finder, for example, and you're standing one foot away from the wall-- now a perfect sensor would always say, you're one foot away from the wall. You're one foot away from the wall. Every single reading is constant. But realistically, there might be Gaussian noise, for example. Or at a more extreme case, it says, your one foot away from the wall. You're two feet. You're right there. There's this distribution.

And so you would ideally characterize this distribution. And you plug that into this model and that formulates your POMDP. This sounds really hairy, but if you work through just a sample iteration of living in a POMDP world, that's not too bad. You start at some state, S. You take an action, A. With some probability, you're going to end up in a bunch of different states based on your transition model. At that point, we can use the lessons we learned from MDP land where we said, when we make observations, we reduce our uncertainty. we collapse into a single state.

So we say, let's make an observation. But this time, observations aren't guaranteed to resolve all our uncertainty. So we make an observation. And that observation is probabilistic based on our current state and the action we just took. And again, obviously, it depends on your current state. Because if you're one foot away from a wall, hopefully you'll get a different characterization of observations than if you're 20 feet away from the wall. Otherwise, your sensor it's totally useless. Are there any questions about this formulation?

AUDIENCE: Yep. Quick question. So we will take observation, any other observation and then you try to infer which state you're in, is it just a clustering problem? For instance, the multi-cluster Gaussian [INAUDIBLE] models. So class A, class B, class C, which are state, then taking an observation there's a high probability [INAUDIBLE] This is what we're tyring to do for each observation over here. So we're trying to find clustering [INAUDIBLE].

PROFESSOR 4: So you could, I imagine, implement an algorithm where, yeah, every time you make an observation, you then try to say, all right. What's my most likely estimate, or maybe my [INAUDIBLE] least cost estimate? But inherent with that is the risk that you're discarding a lot of information, right? Because you're going to generate a probability distribution over your

state. And so, yes, you can say, I'm going to stick with the maximum likelihood estimate. But if you can, you should probably try to maintain that distribution as long as possible. OK. And we'll see that this is really computationally expensive unless you start making some assumptions.

And in the case study we're going to look into, that's exactly what [INAUDIBLE]. But I've seen a lot in the literature that as much as you can, you want to maintain these distributions for improved accuracy. Any other questions? All right. Well, we're going to compare now the execution of a POMDP to the execution of an MDP. We started out-- we're living in the same real world. We've got our same transition model. Everything is peachy. We take action one, and we want to go North. We have this distribution that we generate the states.

And at this point, what did we say? We said, we hate the fact that we have to deal with three cases. Three is two too many. So let's make an observation to collapse this distribution. Now I've described a lot about noisy sensors, right, where basically it's a true measurement plus some noise, maybe Gaussian distribution. There's another partially observable sensor you can have in the POMDP which really feeds into the name, Partially Observable. What if you can only observe part of your state?

For example, if you're living in an x-y grid, maybe you can only observe your y dimension. This match up, in a real world example, to quadrotor flying down a hallway. Catherine was working on a DARPA project with quadrotors flying down hallways. If the hallway is too long, your laser range finder isn't going to be able to determine where you are along one axis. But it can tell where you are along another. And so that's what I've said. I've said, pretend this quadrotor has that sensor. We can only observe its y component. And it says, my y component is 3. I have no idea what my x component is.

Well, this sucks. Right? Because we got rid of this state, but then we couldn't decide, are we at state (1,3) or (3,3)? There's no way of resolving this. So we can re-normalize. We can add in the effect from the observation probabilities saying, maybe, in fact, I'm far more likely to observe a y component of 3 if I'm at (1,3). But in this case, we say it's equally likely to make that observation for those two states.

And so now we've got to deal with these two cases. And so we can take our next action. Instead of resetting to a single state, we've got to keep growing this tree. And what's the key difference between this and when we were executing our MDP? Except we didn't manage to collapse back to a single state. We didn't manage to reset the problem. And this is annoying.

Because you can't execute a policy and say, I'm certain that this is my configuration. So your policy can't map from exact states to actions, because you never know your exact state. Does this make sense? Has everyone lost hope in planning, right? Yeah.

AUDIENCE: So in here because-- so from the left to the right, you're basically mapping from one belief state to another belief state. So it's like a one arrow thing. But and then from that second layer, you should have two arrows. One with probably 0.5 that observes 3, and it gives rise to that belief state it just showed. In one, with probability of 0.5 where the sensor is being 4. Because if you have a 0 probability of looking at (2,4), you might as well just observe 4 as well as your y. So in this case, I'm not sure if you were just trying to show one branch of your POMDP planning, but basically what you have to do is you would go like this to the second layer. You would have a branch with a 0.5 probability on either. One giving you 3, one giving you 4. For the one that is showed, you've got that relief state. For the other one, you get the state (2,4) with probably 1.

PROFESSOR 4: Yeah. So you were perfectly describing planning, right? I should have made this more clear. This isn't planning. This is executing. We have this policy that we're going to execute. And so if we were planning, we would have to consider all these branches and say, well, yeah. There's a 50% chance here I'll end up here, in which case, my y value is going to read 4 and you have to grow this whole thing. I'm saying, no. This is real time execution. yeah. Great question. Any others?

Well, this is a great time to transition to, well, we can't just magically be handed these policies. How do we actually generate them? How do we start planning in the belief space? The belief space is the space distributions of possible configurations. So I'm going to talk about a general class of algorithms. A lot of planners in POMDP land and the belief space plan with probabilistic roadmaps- PRMs. The goal is to generate a policy that maps from a belief state to an action. And I'm going to go into a little more what a belief state is.

But the general algorithm in this graphic illustrates these four steps. We're going to sample points from the configuration space, as if everything was deterministic. We're going to connect those points to nearby points. And define nearby however you want. It could be your closest neighbors, all neighbors within a radius, whatever. As long as those edges don't collide with obstacles. Once you've done that, somehow-- and there's some magic in this-- you're going to transform your configured action space probabilistic roadmap to a probabilistic road map in the belief state. Great. Once you've done that, you can just do shortest path depending on

whatever cost function you use.

And what's really cool is you get different paths for when you stay in the configuration space and when you go to the belief space. So the green path, that bottom right figure, seems a lot longer than the red path. And the reason is the green path was planned in the belief space and followed a lot of landmarks that the quadrotor could take measurements off of. So it was really confident about its position whereas the red path is the shorter path that had a higher likelihood of a collision. And we're going to look into that figure more later.

I'm going to segment this algorithm into two parts. The first part-- these first two steps-- it's just probabilistic road maps. Who here has heard about probabilistic road maps? Raise your hand. OK. 50/50. I'm so excited for the 50% who haven't heard. One of the top algorithms, in my opinion. It's really simple, and it's really powerful. Here's basically almost a complete implementation of probabilistic road maps. It's pseudocode so don't copy paste, but it's almost there.

You're going to construct a graph. You're going to add your start goal. Your start configuration being the green dot and then goal configuration, the red dot. And then you're just going to keep sampling notes those from the free space. You say, how about (2,3)? You're going to add that to your graph. You're going to connect that node to a bunch of other nodes nearby. And then you're just going to keep sampling until maybe you have enough nodes or maybe until you have your complete path. Should be happening there.

And then once you've got this whole graph, you can just find the shortest path along that. And there are some really cool results that if you sample in a good way, and so on, asymptotically. As you start sampling more, you're going to asymptotically approach the best path in a completely continuous space. The power of probabilistic road maps and a bunch of randomized algorithms though is that they scale pretty well to high dimensions. So you don't need to actually consider the continuous space. You can just sample [INAUDIBLE]. Are there any questions about probabilistic road maps?

Really cool. If you are interested, and you just heard about PRMs. You probably haven't heard about RRTs. Those are also really cool.

AUDIENCE: Just a quick [INAUDIBLE] question. So for any node, [INAUDIBLE] at this uniform example [INAUDIBLE].

PROFESSOR 4: Sorry. When you're choosing where to place the dot? Or what to connect to?

AUDIENCE: [INAUDIBLE] there's a dot there on the side [INAUDIBLE] right? So when I do the sampling, I just [INAUDIBLE] this node. I uniformly choose one of them [INAUDIBLE].

PROFESSOR 4: So in general, probabilistic roadmaps, you can throw in whatever sampler you want. The way this particular one-- the way I implement this is you sample points uniformly from the entire space. If it's inside an obstacle and you remove it, once you place it-- this was connect to the K closest-- I think K is 7 in this case. And if there's an edge that if that edge collides with an obstacle, remove it. I'd be happy to go into more detail for that PRMs later.

All right. But that's not enough. Because what we described as PRMs in the configuration space, but what we need to do is somehow elevate a PRM from the configuration space to a belief space. And this is really hard. We don't have access to these raw configurations. Let's imagine we were in this really simple world where the quadrotor could be in three possible states. One, two, three. Really easy, right? Sample a bunch of points. They're going to end up in one, two, or three. You could pretty quickly cover the entire space. But this simple configuration space will transform to the belief space becomes infinite.

You have infinite possible distributions to consider. There is the distribution where you have 100% chance probability-- 100% probability that you're in state 1, 100% probability that you're in state 2. 100% probability that you're in state 3. And then everything in between. We went from three to infinite. This is not boding well. And even if you start saying, well, I'm not going to consider the whole distribution. I just care about the mean and the variance, it's still not pretty, right? Well, this is where we have to start making approximations And this is where you start getting differences in POMDP planners-- where they make assumptions, where they make approximations.

So these images are from the belief road map paper from the Robust Robotics group a couple of years ago. But I'm going to talk about a different planner soon. But the idea behind a lot of these planners is maybe we can start saying what these distributions are going to look like based on our models. So to our planning problem, if we know we started at (2,3) and we know our transition distribution, we can start saying, well, this is my probability distribution. And then when I make and observation, I can build distributions off that.

And so, if you could exhaustively propagate these distributions forward, that would be great. But it's unrealistic. And I just want to point in terms of the visual way to represent these

distributions. A really nice way of saying, in the deterministic world, you have these dots and edges. In the probabilistic world, these circles, these ellipsoids, represent uncertainty. Typically, it's the one standard deviation or three standard deviations away. And so you can start building into the map, these are the distributions and the variances I can see in these nodes. Are there any questions about this stuff.

All right. Well, we're going to delve now into a specific case study. Feedback Based Information State Roadmaps-- FIRM. From now on, that's the only way I'm going to refer to it. The idea behind this is you're going to sample mean configurations from your configuration space. Then you want to build an LQR-- that's Linear Quadratic Regulator controller-- around these mean points. And that will generate what variance you can tolerate. So LQR controllers, if you don't know, they're really nice. Around a small region around a point, they can drive a quadrotor, for example, to that figure.

And so if you build these LQR controllers around points, you can say, all right, anytime I end up in this cloud in the belief space-- so any sort of distribution and can bring it back to that mean. And so now, what we've done is we've generated every point, we just need to connect them. And the idea is if you have a feedback based, they can get from one cloud to another anywhere in the clouds. Then you can get from point to point. All right.

This is how you generate the graph. And what's cool is the way they formulated the problem is they said, well, set up the cost of executing an edge. We switched from rewards to costs because we're pessimists now. Well, the cost is going to be a linear combination of the expected time to execute that edge and the uncertainty along that edge.

Now, this is actually really cool to play with. What do you think would happen if I set beta to 0 in this? Like, what sort of [INAUDIBLE] would you get? I will cold call because I know a few names here. Anybody? I don't want to cold call someone who may [INAUDIBLE].

AUDIENCE: You're going to get the shortest path.

PROFESSOR 4: Shortest path, that's right, right? This turn goes away. The cost is a function of the time-- shortest path. Now, what's cool is one day I was messing around with this code. I'm like, I wonder what happens if I set alpha to 0, right?

So your cost is purely a function of uncertainty. In turns out what the quadrotor does it just hangs out where it starts. It says, I'm in no hurry. I know where I am. I'm just going to stay

here.

But I find this amazing, right? Because this almost models behavior even. You could start saying, do I want to be a risky quadrotor or a safe one? Like, how important it is for me to get somewhere on time or be safe? And it's just those two parameters. Or you could even make it 1, like alpha and 1 minus alpha.

I think this stuff is really cool. The one detail that I'm really going to into for FIRM is the cost equation, which is based on the Bellman backup equation that we had. The cost to go, right, the expected cost from a belief state [INAUDIBLE] is-- well, you're going to take the best action. So you're going to take the mide of this whole thing.

You're going to say, it's the cost of executing a specific action plus the cost of colliding with something, an obstacle, times the probability of colliding, right? That's this term. And then you've got to say, well, OK, and then once I reach a state, what's the cost from there? And then I could weight that by the probability of ending up in that state.

Does this equation make sense to people? There are a lot of symbols, and honestly, I hate notation, but it works. You just plug in-- the cost is I'm going to take the best action. It's going to be the cost of using that action, the cost of colliding times the probability of colliding given that I used that action and I started where I started, and then the cost from where I end up. And since you end up in a probability distribution, we need to consider all these cases. Yeah?

AUDIENCE: When you define like an action from one place to another, do you always think of it as starting from mean that you sampled or from anywhere [INAUDIBLE]?

PROFESSOR 4: So it's from the-- so formally, it was from the belief state, which is the mean, yeah, plus the variance once it stabilized to that point. And the way that variance is generated, I should have said, is, you're going to have models of these quadrotors. And so I spent a good time in the ACL with John Howell, in Course 16. And you just like let the quadrotor hover. You measure its position for a long time. And you get a distribution over where it goes.

AUDIENCE: So what does the letter M stand for?

PROFESSOR 4: What is the letter M?

AUDIENCE: Yeah.

PROFESSOR 4: Right. So you're summing over-- these are the belief states that you could end up in, right? So if everything were deterministic, this would just be ignore the sum. It's where you end up. Realistically you could end up in some other state we haven't considered.

AUDIENCE: So just a quick question. So those, if we're operating on a Gaussian space, then you have Gaussian observations. So those are sums over the observation samples that are generated when [INAUDIBLE]. So that is a finite sum over the possible infinite observation states we might have.

PROFESSOR 4: Yeah. So there's definitely [INAUDIBLE] in terms of where you could end up. And even the action space, there's a set feedback controller that you're allowed. I think the observation, if you modeled it as a Gaussian, if you made some nice assumptions, it can be tractable as continuous.

AUDIENCE: Oh, yeah. I was [INAUDIBLE]. So for the start, so if you start with Gaussian noise, then you have a linear model. Then your prediction's going to be Gaussian.

PROFESSOR 4: Yeah.

AUDIENCE: But then you have Gaussian observations, which is great because then your update is Gaussian but the observation space is infinite. So basically not only you have two sample positions, but you also have two sample potential observations that you might get as you go along.

PROFESSOR 4: Yeah.

AUDIENCE: So that you can basically-- and it's great. But you end up basically reducing a possibly infinite branching, which is your Gaussian to kind of a like a Monte Carlo Tree search. You generate a whole bunch of meaningful samples, and those are the ones that you consider. So is that where the sum is coming from?

PROFESSOR 4: Yeah, basically. And a fun fact, the theorem that I read and trusted was that if you just sample randomly right from your configuration space, you have zero probability of constructing a graph that without any assumptions will be connected. Whereas for PRMs, you sample enough and things will turn out nicely, not the case with the belief state. That's why we need to make these assumptions.

We're killing it. We know POMDPs. Now we get to look at some really fun graphics generated from real flights using FIRM. The big takeaway is that FIRM prefers safer paths. We've got two images that look really similar. We're going to talk about one and then show why they're slightly different.

The test flight that we put this quadrotor under was we said, we're going to start at this configuration. We're going to go this configuration. And there's this big, blue obstacle. And there are landmarks that you can take measurements off of. So those are these red dots.

We want to compare two planners right, a PRM and a FIRM planner. The PRM planner said, right, I just want to minimize time. And so it found the first path is stay to the left of the obstacle. But that's actually a really narrow path between the obstacle and the wall.

On the other hand, the FIRM planner said, right, I want to minimize the linear combination of time and uncertainty. And so if you ramp up the term the wait for uncertainty, at some point the path sort of pops over the obstacle. It's really cool. It snaps. And then you get this safer path. But that's longer. And how do we know it's safer? Because these ellipsoids that we've drawn are the 3D versions of what we saw for PRM elevated to the belief space version.

It's the uncertainty that the quadrotor has over its true state. Why is the PRM plan-- why does it have such big ellipsoids? It's because when it's behind the obstacle, it can't make any of these measurements because it can't see the landmarks. So it's basically using dead reckoning. And the transition model, right, is not deterministic. So it's uncertainty grows. And so we can see these ellipsoids are bigger for PRM than FIRM. And then the reason I have two images is they are slightly different.

It's that these landmarks were fictitious landmarks that we just said, you can-- and we would generate fake measurements off of them. And so we could tune the noise of the landmarks. Maybe a little bit of cheating, but it allowed us to say, would it increase the noise from some-- the dimension was number 0.05 to 0.15.

You can see the uncertainty ellipsoids from PRM grow when we increased the noise, whereas for FIRM, they stay about the same. And importantly, these ellipsoids grow enough that they start overlapping with the obstacles. That represents a very high probability of collision.

Whereas the FIRM, the way it managed to keep these ellipsoids so small is we kept the cost function the same, right-- the same weight on uncertainty, same weight on the time, right? But

the uncertainty was so much higher. And so we decided, well, I can sacrifice a little bit of time.

I can take the slower path. I can just hang out by landmark, really make sure I know where I am before continuing. And so the path-- the duration of the flight took a lot longer for FIRM as time would increase but the uncertainty would stay about constant. Do people understand these graphics? Great. All right.

We've got a graph that just represents a little more formally that growing uncertainly. As noise increases, the variance along a single dimension, z, y, to x, for PRM, which is that, and FIRM. The variance for PRM is always higher, and it grows. For FIRM, it's lower, and stays about constant. That's the big takeaway. FIRM minimizes this uncertainty. And then the final image from these results that I want to show is in simulation.

They said, well, let's actually measure how often it crashes. We didn't want to do this in the real world because we don't want to crash the quadrotor that many times. The gist of it is comparing a reactive planner and a deterministic one [INAUDIBLE]. As noise increases-- noise was simulated with wind strength-- the number of crashes increases for PRM, basically. And for FIRM, it stays constant and low. The reason there are two lines is there were two planners with different time horizons. The important thing is FIRM is low and constant. PRM grows.

We've talked now throughout all probabilistic planning you ever need to know, right? No, not quite. But we have covered a lot of stuff. What are the big takeaways?

We've learned that real-world problems are stochastic, right? Quadrotors are not these perfect machines that we wish they were. But it's important to model them as stochastic. The problem is once you start modeling them as stochastic, it becomes a lot harder to solve.

But if you make some assumptions, or even if you don't, if you're just get smart and you do the heuristics, you can resolve this complexity. And so I hope you remember these three points, you remember this graphic or that graphic, the idea that if you take uncertainty into account, you get fundamentally different paths [INAUDIBLE]. And that can be a good thing.

What questions do you have, anything? Yeah?

AUDIENCE: [INAUDIBLE] so far people have [INAUDIBLE] problems. So how do the same, you know, [INAUDIBLE]? So for instance, we suggest that this is the maximum risk we want to take. Is it possible to integrate each of our constraints into your optimization problem and solve it? Or [INAUDIBLE]?

PROFESSOR 4: So I imagine one thing that I would like to test is if we go back to this really crude version of our cost equation, we can imagine saying that if we want to come up with a bound for uncertainty that we can tolerate, you could maybe like setting an intercept for this to be that bound and then just ramping this up.

AUDIENCE: Oh, [INAUDIBLE] multiplier [INAUDIBLE].

PROFESSOR 4: Something like that.

AUDIENCE: [INAUDIBLE].

GUEST SPEAKER: Yeah, exactly. So it only comes into effect. Possibly what I said is a terrible idea. But it's-- especially in simulation, you can just try it out and see if it works.

AUDIENCE: And just another question, you said propagated probability solution on the networks is hard. [INAUDIBLE]. So therefore we can assume an illustrated area, and then [INAUDIBLE] normal distribution, they are contrary to each other. Therefore you can [INAUDIBLE] observation and then updating [INAUDIBLE] and form a distribution, right?

PROFESSOR 4: Sure. So I think that might be making some assumptions that we might not be willing to make about the nature of the distributions that you're trying to propagate. I need think about this some more. But I think intuitively, we can understand that any time you're propagating a distribution versus a single discrete value, it's definitely not going to be easier. And so as your distributions become more complex. Perhaps if you're modeling a real-world stochastic sensor, you might not be able to perform these efficient updates using conjugate priority. Any other questions? Otherwise we-- yeah?

AUDIENCE: So all these FIRM examples you're showing, this is all planning done offline. [INAUDIBLE] offline, right? Have you expanded this at all to like an online case? Or does that all require re-planning? And how long does it take?

PROFESSOR 4: Right. So it's not good, I can tell you that much. What's nice is FIRM generates a policy. So if you construct your PRM to say don't just find the shortest path, keep sampling points until you're confident that you're always going to end up near some point. You can construct a policy and then just online look up what's my belief that matches most closely.

It took for-- I think we sampled 600 nodes in like-- what are the dimensions on this? So 2 to 3

meter by 8 by 3 meter thing-- it took about 15 minutes. It was really slow. Now, granted it was a virtual machine. But it's not something we want to re-plan on the fly. With PRM, typically a lot faster.

AUDIENCE: Who was a lot faster in that case?

PROFESSOR 4: So when we just did PRM, I think it was one minute or something, at least an order of magnitude. And you could use other planners as well, like RRTs. And there are RRT versions of-- there are POMDT versions of RRTs. But [INAUDIBLE]. All right, I'm going to turn it over to Steve so we can learn about this project.

[APPLAUSE]

STEVE: So I'm just going to say a few good words about the Grand Challenge. So again, the final assignment will be released tonight for this. And as Professor Warrens mentioned, it's going to be descoped a bit from what we originally had in the syllabus due to time constraints. But here's a preview of what you'll be doing.

So for an overview, it will be a class-wide collaboration. So what we're going to do, you guys are going to stay in your advanced lecture teams and each basically apply what you've done, the great work you've done on that, onto our hardwork robot that we have. so it's not a competition where each team will be competing against each other. And so again, we're descoping it. The syllabus originally said it was 20% of your grade. It's probably going to be more like 10% or 15% in the end.

So this is the robot you guys will be using. It's called an [INAUDIBLE] robot. Usually it has sort of a robotic arm on it. But we're actually not going to be using it for this [INAUDIBLE]. This base made from a company from Spain, Orbonix.

It's a pretty cool robot. One cool thing about it is that it's actually omnidirectional. So there are wheels on your wheels here. And what that means is that it can drive sideways, left to right. It would make parallel parking your car very easy.

[LAUGHTER]

We're not going to be driving it probably this fast because it's actually super heavy. And we don't want anyone to-- yeah, we forgot to screw in that [INAUDIBLE].

[LAUGHTER]

It will be screwed in during the competition. But it's a pretty fun robot. So you guys will be working on this. And so the actual challenge itself, as we've mentioned many times throughout this class, will be a modified orienteering your challenge.

So there's going to be a few different challenge stations that you have to drive to. And at the stations, there'll be small computational challenges. And those computational challenges will be a subset of the advanced lecture teams, not all of them. And the goal is to complete as many of those as you can and try to do that as quickly as possible. And it's also going to be held indoors in our lab space, where we just showed you. So that way if it rains, we can still have the Grand Challenge at the end of the semester.

So this is sort of how it's set up as of last night actually. So basically the robot will be abe to drive around in a small little LEGO maze that we set up. And there's going to be sort of different things that you have to do with in different places.

So what are you actually going to be doing? What assignment are you guys going to be doing? Well, it's actually a little flexible. Since each of you did different things for your advanced lectures, each team is going to have a bit of a different assignment applied to this.

I have some proposed ideas that I'm going to talk about in the next slide here. But the big thing is that these are just ideas. You guys have a lot of flexibility in this. You'll be probably working with the [INAUDIBLE] a lot to have access to the robot.

We can arrange extra office hours for you guys to come use the hardware to test things. So we'll be arranging all of that things as sort of on an as-needed basis. If you want to-- basically it'll sort of depend on your team.

AUDIENCE: And the people who'll be helping out are you, us--

STEVE: Yes, me and Tiago, who gave a lecture earlier in the semester and possibly also a few other people from our lab. But we'll be the main contact points for it. So of course it should go without saying, but all the team members should contribute equally within your team.

So it'll be maybe less structure than in advanced lecture. But just make sure that everyone's contributing equally in the assignment. And it's going to involve using this thing called the Robot Operating System, or ROS, which is basically a software framework for communicating

and it's used a lot in robotics.

Just a quick show of hands, how many of you have used ROS before? Oh, wow, so a lot of used ROS. How many have heard of ROS, if not used it? OK, so a lot of people. So that's a good starting point.

So here are the the proposed tasks for each group. And of course, all of these are up for change. If you guys want to change it, let me know. Basically, so some of the groups, it's very clear how it immediately applied to the Grand Challenge. So incremental path planning-- well, we have a mobile robot, so maybe we can actually print that on the robot and get it to change or plan if something gets in the way.

The semantic globalization group-- obviously very applicable to the Grand Challenge. You need to know where you are. So the robot that we have now can actually do normal metric localization. So it can sort of know where it is.

But what will be interesting to see is compare the semantic localization to that one. But how would you do semantic localization? Well, we can use a camera. And we can choose the visual learning through deep classification-- the visual classification through deep learning group. So I think these two groups would have a really nice synergy and a really cool way to work together.

So the MCTS group are very cool, but I had a little trouble thinking about exactly how that would apply. So maybe you look like you have an idea maybe.

AUDIENCE: So to clarify, all these are separate runs of the robot?

STEVE: So we're actually probably going to run all of them-- so the grid is-- this would be probably one run of the robot. We're going to decouple these so that if one or a subset these don't work super well, the other groups will still be able to run. So we're carefully planning that out too. So the MCTS group, I was thinking maybe could solve-- that could be a really nice way to-- one of those computational challenges.

Maybe you have to play against a human. And if it wins, great. You can go faster or you get more points. So you could implement on a different game other than connect four or possibly and sort of [wrap it around [? rod ?] that to call with that.

So each ability group I think would also be a great place to do one of these challenge stations.

So we could give you guys puzzle, say maybe it's some sort of maze-like state space. And you have to see could we even the reach the goal here? And if you get the answer right, well, you an move on to the next stage or get more points or something like that.

AUDIENCE: But again, these are just suggestions. So for example, they can do the reachability for doing motion planning.

STEVE: Yeah, so if you guys have other suggestions on how to implement your team's stuff into the Grand Challenge, definitely send me an email, preferably today or as soon as you think of the things. And we can change these. These are just suggestions for right now.

The last two are sort of more planning related. So planning with temporal logic, which was Monday's lecture, I thought would be cool way to sort of control the robot's high-level action. So maybe that could involve modeling are Grand Challenge with PDDL. And maybe with linear temporal logic goals, models that you get [INAUDIBLE]. Then you could compile that and call up to turn around it and execute that plan on the actual robot, do [INAUDIBLE] high-level actions of the robot. That's a possibility. And for today's group, the infinite horizon probablistics planning, maybe you could do something actually similar.

But instead of modeling the domain as PDDL model it as an NVP and solve it with LAO star, and get sort of a policy on how to control the robot. Maybe we can change it a little bit so that certain squares can be more risky than others or so on. So again, there's flexibility in all of these here.

So these are just some of the suggestions. Does anyone have any questions before we-- this all I have. So anyone have any questions? Yeah?

AUDIENCE: How are we working on [INAUDIBLE]? Are you [INAUDIBLE]?

STEVE: So it's basically up to you guys really divide up the work amongst yourselves. It's really different for every team. So we're--

AUDIENCE: Sorry. So each team is developing [INAUDIBLE].

STEVE: Right, yeah. Each team is really in their own separate package FIRM. For example, these two teams, there's probably going to be a common interface where the output of this one goes to the input of that one. So for that one, we're going to give you sort of the interface [INAUDIBLE]

message type package for that. But other than than, like for dividing up the work, you guys [INAUDIBLE] that.

PROFESSOR 2: So your plans will be integration if you're trying [INAUDIBLE] pieces the group is doing. That we think is, thing that uses soft engineering skills [INAUDIBLE]. It can be really unpleasant to do, take a lot of time. So we don't want you to have that experience. So that's why you can... If not only what we wanted you to do is to be able to get your own capability [INAUDIBLE]. If you guys choose to integrate with other teams because you're really excited about that and because it looks like the people that you're working for [INAUDIBLE]. Then that's purely your choice. Is that fair enough?

STEVE: Sure. It's fine. Any more questions? OK. Sounds great.

[APPLAUSE]