MachineLearning-Lecture19

Instructor (Andrew Ng):All right, good morning. So just one administrative thing before we go into today's technical material. So let's see. The final projects poster session presentations will be on Wednesday the 12th of December. I guess that's one week from this Wednesday. So similar to the mid-term exam, if you physically live in the Bay Area then please come in person to do the poster session. So this means you guys, you know, so if on campus students, and SCPD students are that, so they live in the Bay Area. If, for some reason, you live in the Bay Area, but aren't able to come in in person to the poster session, please send us an e-mail as soon as possible at the usual class mailing address cs229qa@cscenter.edu; it's the one on the course website; let us know. And if you're an SCPD or SITS student and if you live physically outside the Bay Area then you're exempt from the poster session, I guess. The poster session itself will be held at 8:30 a.m. on this – so the registrar assigns classes a certain time slot for final exams, so this class doesn't have a written final. We use that time slot instead for the final poster presentations. So they're about, I don't know, I think there are, like, about 70 posters in this class. So in order for me to be able to see everything in a reasonable amount of time please prepare, like, about a two-minute presentation, so that as I go around the posters next Wednesday you can just briefly tell me about your work. And that's the short, and then as I promise every year, I personally read every single word of every single final project write-up, so don't worry about telling me everything, every little detail, of what you did because when you send me your final project write-up – so I promise I read every single word of every write-up, so I get all the details from there. But to go in the process presentation is actually first and foremost for you guys also to have a look at what other cool things people in the class are doing and based on the models, I've been flipping through the models, of all the very cool projects this year. So at the poster presentation too hopefully you can see what your colleagues, your classmates, have been doing and have fun doing that and also for me to just get a very brief sense of the things you've been doing. Okay?

The posters themselves, we will supply poster boards if you want them. You're welcome to buy your own poster boards, but in a few days I'll send instructions on where you can pick up a poster board. If you do get a poster board from us we do reuse them. We do recycle them from year to year, so if you do get one from us that I ask you to return it at the end of the poster session next Wednesday. So poster boards are, I don't know, maybe about this wide and about this tall, so you can print out several pieces of paper. Print out several slides and attach them to the poster board. And we'll provide easels, as well, for the poster presentation. Okay? So next Wednesday, come a little bit before 8:30 a.m. and we'll meet on the first floor of the Gates lobby. So around the first floor of the Gates lobby around Gates 104 and we'll set up poster there. Okay? Are there questions about the poster session? No? Okay, cool. Let's see.

Okay. So welcome back, and let's continue our discussion of reinforcement learning algorithms. On for today, the first thing I want to do is actually talk a little bit about debugging reinforcement learning algorithms and then I'll continue the technical discussion from last week on LQR, on linear-quadratic regulation. In particular, I want to

tell you about an algorithm called the French dynamic programming, which I think is actually a very effective absolute controls lack reinforcement learning algorithm for many problems. Then we'll talk about Kalman filters and linear-quadratic Gaussian control, LGG. Let's start with debugging RL algorithms. And can I switch to the laptop display, please?

And so this was actually – what I'm about to do here is this is actually a specific example that I had done earlier in this quarter, but that I promised to do again. So remember, you know, what was it? Roughly halfway through the quarter I'd given a lecture on debugging learning algorithms, right? This idea that very often you run a learning algorithm and it, you know, maybe does roughly what you want to and maybe it doesn't do as well as you're hoping. Then what do you do next? And the talk of this idea that, you know, some of the really, really good people in machine learning, the people that really understand learning algorithms, they're really good at getting these things to work. Very often what they're really good at is at figuring out why a learning algorithm is working or is not working and that prevents them from doing things for six months that someone else may be able to just look at and say gee, there was no point collecting more training data, because your learning algorithm had high bias rather than high variance. So that six months you spent collecting more training data – I could have told you six months ago it was a waste of time. Right?

So these are sorts of things that some of the people are really good at machine learning, that they really get machine learning, are very good at. Well, just a few of my slides. These slides I won't actually talk about these, but these are exactly the same slides you saw last time. Actually, I'll just skip ahead, I guess. So last time you saw this discussion on – right. Diagnostics for whether you happen to have a bias problem, or a variance problem, or in other cases where the – your optimization algorithm is converging or whether it's the [inaudible] optimization objective and so on. And we'll talk about that again, but the one example that I, sort of, promised to show again was actually a reinforcement learning example, but at that time we hadn't talked about reinforcement learning yet, so I promised to do exactly the same example again, all right?

So let's go for the example. The motivating example was robotic control. Let's see. Write a – let's say you want to design a controller for this helicopter. So this is a very typical way by which you might apply machine-learning algorithm or several machine-learning algorithms to a control problem, right? Which is you might first build a simulator – so control problem is you want to build a controller to make the helicopter hover in place, right? So the first thing you want to do is build a simulator of the helicopter. And this just means model of the state transition probabilities, a piece of SA of the helicopter, and you can do this by many different ways. Maybe you can try reading a helicopter textbook and building a simulator based on what's known about aerodynamics of helicopters. It actually turns out this is very hard to do. Another thing you could do is collect data and maybe fit a linear model, or maybe fit a non-linear model, to what the next stage is as a function of current state and current action. Okay? So there's different ways of estimating the state transition probabilities. So, now, you now have a simulator and I'm showing a screen shot of our simulator we have a stand fit on the upper right there. Second thing

you might do is then choose a reward function. So you might choose this sort of quadratic cos function, right? So the reward for being at a state X is going to be minus the norm difference between a current state and some desired state of your one simple example of a reward function. And this, sort of, quadratic reward function is what we've been using in the last lecture in LQR control, linear-quadratic regulation control.

And finally, right? Random reinforcement learning algorithm in simulation, meaning that you use your model of the dynamics to try to maximize that final horizon sum of rewards and when you do that you get a policy out, which I'm gonna call the policy pi subscript RL to denote the policy output by the reinforcement learning algorithm. Okay? Let's say you do this and the resulting controller gets much worse performance than a human pilot that you hire to fly the helicopter for you. So how do you go about figuring out what to do next? Well, actually, you have several things you might do, right? You might try to improve the simulator, so there are exactly three steps. You want say maybe after the new model from the helicopter dynamics, but I think it's non – it is actually non-linear. Or maybe you want to collect more training data, so you can get a better estimates of the transition probabilities of the helicopter. Or maybe you want to fiddle with the features you used to model the dynamics of your helicopter, right? Other things you might do is, you might modify the reward function R if you think, you know, it's not just a quadratic function, maybe it's something else. I don't know. Or maybe you're unsatisfied with the reinforcement learning algorithm. Maybe you think, you know, the algorithm isn't quite doing the right thing. Or maybe you think you need to discritize the states more finely in order to apply your reinforcement learning algorithm. Or maybe you need to fiddle with the features you used in value function approximations. Okay? So these are three examples of things you might do and, again, quite often if you chose the wrong one to work on you can easily spend, you know – actually this one I don't want to say six months. You can easily spend a year or two working on the wrong thing. Hey, Dan, this is a favor. I'm, sort of, out of chalk could you wander around and help me get? Thanks.

So the team does three things; they'll copy the yellow box to the upper right of this slide. What can you do? So this is the, sort of, reasoning we actually go through on the helicopter project often and to decide what to work on. So let me just step for this example fairly slowly. So this, sort of, reasoning you might go through. Suppose these three assumptions hold true, right? Suppose that the helicopter simulator is accurate, so let's suppose that you built an accurate model of the dynamics. And suppose that, sort of, I turned two under slides, so suppose that the reinforcement learning algorithm correctly controls the helicopter in simulation. So it's a maximized that expected payoff, right? And suppose that maximizing the expected payoff corresponds to autonomous flight, right? If all of these three assumptions holds true, then that means that you would expect the learned controller pi subscript RL to fly well on the actual helicopter. Okay? So this is the – I'm, sort of, showing you the source of the reasoning that I go through when I'm trying to come up with a set of diagnostics for this problem. So these are some of the diagnostics we actually use routinely on various revised control problems. So pi subscript RL, right? We said it doesn't fly well on the actual helicopter. So the first diagnostic you want to run is just check if it flies well in simulation, all right? So if it flies well in simulation, but not in real life, then the problem is in the simulator, right? Because the

simulator predicts that your controller, the pi subscript RL, flies well, but it doesn't actually fly well in real life. So if this holds true, then that suggests a problem is in the simulator. Question?

Student:[Inaudible] the helicopter pilots try to fly on the simulator? Do you have the real helicopter pilots try to fly on the simulator?

Instructor (Andrew Ng):Do I try to have the real helicopter flying simulator?

Student: Real helicopter pilots.

Instructor (Andrew Ng):Oh, I see. Do we ask the helicopter pilots to fly in simulation? So, yeah. It turns out one of the later diagnostics you could do that. On our project we don't do that very often. We informally ask the pilot, who's one of the best pilots in the world, Gary Zoco, to look at the simulator sometimes. We don't very often ask him to fly in the simulator. That answers your question. But let me actually go on and show you some of the other diagnostics that you might use then. Okay? Second is let me use pi subscript human to denote the human control policy, right? This is pi subscript human is policy that, you know, however the human flies it. So one thing to do is look at the value of pi RL compared to the value of pi subscript human. Okay? So what this means really is, look at how the helicopter looks like when it's flying under control of the pi subscript RL and look at what the helicopter does when it's flying under the human pilot control and evaluate – and then, sort of, compute the sum of rewards, right? For your human pilot performance and compute the sum of rewards for the learning controller performance and see on, say, the real helicopter or that question or you can do this on the real helicopter or on simulation actually, but you can see does the human obtain a higher or a lower sum of rewards on average than does the controller you just learned. Okay? And then the way you do this you actually can go and fly the helicopter and just measure the sum of rewards, right? On the actual sequence of states the helicopter flew through, right? So if this condition holds true, right? Where my mouse pointer is if – oh, excuse me. Okay. If this condition holds true where my mouse pointer is, if it holds true that pi subscript RL is less than pi subscript human, those of you watching online I don't know if you can see this, but this V pi subscript RL of as zero less than V pi subscript human of as zero. But if this holds true, then that suggests that a problem is in the reinforcement learning algorithm because the human has found a policy that obtains a higher reward than does the reinforcement learning algorithm. So this proves, or this shows, that your reinforcement learning algorithm is not maximizing the sum of rewards, right? And lastly, the last condition is this – the last test is this, if the inequality holds in the opposite direction – so if the reinforcement learning algorithm obtains a higher sum of rewards on average than does the human, but the reinforcement learning algorithm still flies worse than the human does, right? Then this suggests that the problem is in the cos functions than the reward function because the reinforcement learning algorithm is obtaining a higher sum of rewards than the human, but it still flies worse than the human. So that suggests that maximizing the sum of rewards does not correspond to very good autonomous flight. So if this holds true, then the problem is in your reward function. Or in other words, the problem is in your optimization objective and then rather than the

algorithm that's trying to maximize the optimization objective and, so you might change your optimization objective. In other words, you might change reward function. Okay?

So, of course, this is just one example of how you might debug a reinforcement learning algorithm. And these particular set of diagnostics happen to apply only because we're fortunate enough to be able to an amazingly good human pilot who can fly a helicopter for us, right? If you didn't have a very good pilot then maybe some of these diagnostics won't apply and you'll have to come up with other ones. But want to go through this again as an example of the source of diagnostics you use to debug a reinforcement learning algorithm. And the point of this example isn't so much that I want you to remember the specifics of the diagnostics, right? The point of this is really for your own problem, be it supervised learning, unsupervised learning, reinforcement learning, whatever. You very often have to come up with your own diagnostics, your own debugging tools, to figure out why an algorithm is working and why an algorithm isn't working. And this is one example of what we actually do on the helicopter. Okay? Questions about this? Yeah, Justin?

Student:I'm just curious how do you collect, like, training data? Like in our homework the pendulum fell over lots of times, but how do you work that with an expensive helicopter?

Instructor (Andrew Ng):Yeah. So, I see, right. So on the helicopter the way we collect data to learn [inaudible] and improbabilities is we usually – not – done lots of things, but to first approximation, the most basic merger is if you ask a human pilot to just fly the helicopter around for you and he's not gonna crash it. So you can collect lots of data as is being controlled by a human pilot. And, as I say, it turns out in data collection the, sort of, a few standard ways to collect data are to let you – so a helicopter can do lots of things and you don't want to collect data representing only one small part of the flight regime. So concretely what we often do is ask the pilot to carry out frequency sweeps and what that means, very informally, is you imagine them holding a control stick, right? In their hand. Frequency sweeps are a process where you start off making very slow oscillations and then you start taking your control stick and moving it back and forth faster and faster, so you sort of sweep out the range of frequencies ranging from very slow slightly [inaudible] by oscillations until you go faster and faster and you're sort of directing the control seat back and forth. So this is – oh, good. Thank you. So that's, sort of, one standard way that we use to collect data on various robotics. It may or may not apply to different robots or to different systems you work on. And, as I say, in reality we do a lot of things. Sometimes we have a controllers collect data autonomously too, but that's other more complex algorithms. Anything else?

Student: In the first point the goal communicates that the other is perhaps not mapping the actions similar to the simulator, so in [inaudible] this could be very common that you could have [inaudible]? Just any specific matter that pilots use to take care of that variable, so all [inaudible]?

Instructor (Andrew Ng):Yeah, right. So you're saying like point one may not be simulated accurate; it may be that the hardware is doing something strange with the control. There is concluding the controls through some action and through some strange transformation because it's actually simulating it as a helicopter. I don't know. Yeah. I've definitely seen that happen on some other robots before. So maybe diagnostic one here is a better form as deciding whether the simulator matches the actual hardware, I don't know. Yeah. That's another class of those to watch out for. If you suspect that's the case, I can't think of a good, sort of, diagnostic right now to confirm that, but that, again, before – that would be a good thing to try to come up with a diagnostic for to see if there might be something wrong with the hardware. And I think these are by no means the definitive diagnostics or anything like that. It's just sort of an example, but it would be great if you come up with other diagnostics to check that the hardware is working properly that would be a great thing to do, too. Okay. Is this – okay, last couple of questions before we move on?

Student: You said the reward function was?

Instructor (Andrew Ng):Oh, in this example, what I was just using a quadratic constant. On the helicopter we often use things that are much more complicated.

Student:[Inaudible] You have no way of knowing what is the, like, desired focus?

Instructor (Andrew Ng):Yeah. So you can, sort of, figure out where – ask the human pilot to hover in place and guess what his desire was. Again, these aren't constants telling you to, yeah.

Student:[Inaudible] physics that are based learning problems. Do you – does it actually work best to use a physical finian model? You know, just, sort of, physics tell or you just sort of do learning on [inaudible]?

Instructor (Andrew Ng):Yeah. The physics models work well, right? So the answer is it varies a lot from problem to problem. It turns out that the aerodynamics of helicopters, I think, aren't understood well enough that you can look at the "specs" of a helicopter and build a very good physics simulator. So on the helicopter, we actually learn the dynamics. And I don't know how to build a physics model. For other problems, like if you actually have an inverted pendulum problem or something, there are many other problems for which the aerodynamics are much better understood and for which physic simulators work perfectly fine, but it depends a lot – on some problems physic simulators work great on some they probably aren't great on all. Okay? Cool.

So, I guess, retract the chalkboard, please. All right. So how much time do I have? Twenty minutes. Okay. So let's go back to our discussion on LQR Control, linear-quadratic regulation control, and then I want to take that a little bit further and tell you about the, sort of, one variation on the LQR called differential dynamic programming. Just to recap, just to remind you of what LQR, or linear-quadratic regulation control, is, in the last lecture I defined a horizon problem where your goal is to maximize, right? Just

sort of find the horizon sum of rewards, and there's no discounting anymore, and then we came up with this dynamic programming algorithm, right? Okay.

Then we came up with this dynamic programming algorithm in the last lecture, where you compute V star of capital T that's one value function for the last time step. So, in other words, what's the value if your star in the state S and you just get to take one action and then the clock runs out. The aerodynamic programming algorithm that we repeatedly compute V star lowercase t in terms of V star t plus one, so we compute V star capital T and then recurs backwards, right? And so on, until we get down to V star zero and then pi star was given by, as usual, the augmax of the thing we had in the definition of the value function.

So last time, the specific example we saw of – or one specific example, that sort of define a horizon problem that we solved by DP was the LQR problem where we worked directly with continuous state and actions. And in the LQR problem we had these linear dynamics where the state ST plus one is a linear function of the previous state and action and then plus this Gaussian noise WT, which is covariance sigma W. And I said briefly last time, that one specific way to come up with these linear dynamics, right? Oh, excuse me. One specific way to take a system and come up with a linear model for it is if you have some simulator, say, right? So in this cartoon, the vertical axis represents ST plus one and the horizontal axis represents STAT. So say you have a simulator, right? And let's say it determines [inaudible] simulator, so we have a [inaudible] simulator that tells you what the next state is ST plus one is a function of previous state and action. And then you can choose a point around which to linearize this simulator, by which I mean that you choose a point, an approximate your – approximate the function F using a linear function, this tangent, to the function F at that point. So if you do that you have ST plus one equals – shoot.

Sorry about writing down so many lines, but this will be the linearization approximation to the function F where I've taken a linearization around the specific point as bar T, A bar T. Okay? And so you can take those and just narrow it down to a linear equation like that where the next state ST plus one is now some linear function of the previous state ST and AT and these matrixes, AT and BT, will depend on your choice of location around which to linearize this function. Okay? I said last time, that this linearization approximation you, sort of, expect to be particularly good in the vicinity of, as bar T, A bar T because this linear function is a pretty good approximation to F, right? So if in this little neighborhood there. And – yes?

Student:[Inaudible] is there an assumption that you are looking at something the second recently indicates like the helicopter, are you assuming pilot behavior is the same as [inaudible] behavior or –

Instructor (Andrew Ng):Yeah, right. So let me not call this as an assumption. Let me just say that when I use this algorithm, when I choose to linearize this way, then my approximation would be physically good in the vicinity here and it may be less good elsewhere and, so let me – when I actually talk about DDP I actually make use of this

property. Which is why I'm going over it now. Okay? But, yeah. There is an intuition that you want to linearize near the vicinity—near of states, so you expect your system to spend the most time. Right.

So, okay. So this is how you might come up with a linear model and if you do that then, oh, you can – let's see. So for LQR we also have this sort of quadratic reward function, right? Where the matrixes UT and VT are positive semi-definite, so the rewards are always negative, that's this minus sign. And then if you take exactly the dynamic programming algorithm, that I've written down just now, then – lets see. It turns out that the value function at every state, excuse me. It turns out the value function for every time step will be a quadratic function of the state and can be written like that. And so you initialize the dynamic programming algorithm as follows. And I just write this down, but there's actually just one property I want to point out later, but this equation is, well, somewhat big and hairy, but don't worry about most of its details.

Let me just put this. Shoot, there's one more equation I want to fit in. Well, okay. All right. So it turns out the value function is a quadratic function where V star T is this and, so you initialize the dynamic programming step with this. This fi and this si gives you V capital T and then it records backwards. So these two equations express – will give you V subscript T as a function of VT plus one. Okay? So it incurs backwards for this learning algorithm. And the last thing, get this on the same board, so – sorry about the disorganized use of the boards. I just wanted this on the same place.

And, finally, the actual policy pi star of ST is given by some linear function of ST, so LT here is a matrix where LT is equal to – numerous times. Okay? And so when you do this you now have the actual policy, right? So just concretely, you run the dynamic programming algorithm to compute fi T and si T for all values of T and then you plug it in to compute the matrixes LT and now you know the optimal action stake and [inaudible]. Okay? So there's one very interesting property – these equations are a huge mess, but you can re-derive them yourself, but don't worry about the details. But this is one specific property of this dynamic programming algorithm that's very interesting that I want to point out. Which is the following. Notice that to compute the optimal policy I need to compute these matrixes LT and notice that LT depends on A, it depends on B, it depends on D, and it depends on fi, but it doesn't depend on si, right? Notice this further that when I carry out my dynamic programming algorithm my recursive definition for si – well it depends on – oh, excuse me. It should be si T, right. Si T plus one. Okay. In order to carry out my dynamic programming algorithm, right? For si T I need to know what si T plus one is. So si T depends on these things, but in order to carry out the dynamic programming for fi T, fi T doesn't actually depend on si T plus one, right? And so in other words in order to compute the fi T's I don't need these si's. So if all I want is the fi's I can actually omit this step of the dynamic programming algorithm and not bother to carry out the dynamic programming algorithm in terms of the fi's. And then having done my dynamic programming algorithm just for – excuse me, I misspoke. You – I can forget about the si's and just do the dynamic programming updates for the fi matrixes and then having done my DP updates for the fi T I can then plug this into this formula to compute LT. Okay?

And so one other thing about it is you can, to be slightly more efficient – efficiency isn't really the issue, but if you want you can actually forget about the fi T's. You actually don't need to compute that at all. Now, the other interesting property of this is that the matrix sigma W appears only in my DV update for the si T's. It doesn't actually appear in my updates for the fi T's. So you remember – well, my model was that ST plus one equals ATST plus VT, AT plus WT where these noise terms, WT, had a covariance sigma W and so the only place that appears – the covariance of the noise terms of appears is in those IT's, but I just said that they can do this entire [inaudible] ordering algorithm without the si T's. So what this means is that you can actually find the optimal policy without knowing what the covariance of the noise terms are. Okay?

So this is a very special property of LQR systems and once you change anything, once you go away from a linear dynamical system, or once you change almost any aspect of this because at discreet states or discreet actions or whatever and once you change almost any aspect of this problem this property will no longer hold true because this is a very special property of LQR systems that the optimal policy does not actually depend on the noise magnitude of these noise terms. Okay? And the only important property is that the noise function of zero mean. So there's this intuition that to compute the optimal policy you can just ignore the noise terms. Or as if, so as long as you know the expected value of your state ST plus one – write down. On average ST plus one is ATST plus BTAT, then there's as if you can ignore the noise in your next state ST plus one. And the optimal policy doesn't change. Okay?

So we'll actually come back to this in a minute. Later on we'll talk about Kalman filters. We'll actually use this property of LQR systems. Just to point out, note that the value function does depend on the noise covariance, right? The value function here does depend on si T. So the larger the noise in your system the worse your value function. So this does depend on the noise, but it's the optimal policy that doesn't depend on the noise. We'll use this property later. Okay. So let's see how we're doing on time. Let's see. Right. Okay. So let's put this aside for now. What I want to do now is tell you about one specific way of applying LQR that's called differential dynamic programming. And as in most of the example, think of try to control a system, like a helicopter or a car or even a chemical plant, with some continuous state. So for the sake of thinking through this example, just imagine trying to control a helicopter. And let's say you have some simulator that espece with the next state is a function of the previous data in action, right? And for this let's say the model of your simulator is non-linear, but and determinalistic. Okay? So I say just now, that the noise terms don't matter very much. So let's just work with the term simulator for now, but let's say F is non-linear. And let's say there's some specific trajectory that you want the helicopter to follow, all right? So I want to talk about how to apply LQR to get helicopter or a car or a chemical plant where your state variables may depend on the amounts of different chemicals and the mixes of chemicals you have in different batch, really. It's really easy to think about a helicopter. Let's say there's some trajectory you want the helicopter to follow. So here's what the differential dynamic programming it does. First step is come up with what I'm gonna call some nominal trajectory, right? And so we're gonna call this S zero A zero. Okay? So one way to come up with this would be if you had some very bad controller – someone hacked a

controller for flying a helicopter is not a good controller at all. But you might then go ahead and fly the helicopter using a very bad, a very sloppy, controller and you get out some sequence of states and actions, right? So I'm gonna – and I just call this sequence of states and actions the trajectory – the nominal trajectory. Then I will linearize F around this normal trajectory. Okay? So i.e., right? I'll use that same thing. So for times si T our approximate ST plus one, as this linearization thing that we just saw, times – plus the other term. Okay? And then you distill this down to sum ATST plus BTST. Okay? So this will actually be the first time that I'll make explicit use of the ability of LQR or these finer horizon problems to handle non-stationery dynamics. In particular, for this example, it will be important that AT and BT depend on time – oh, excuse me. Okay?

So the intuition is that even if this is a pretty sloppy controller, or even if you had a pretty bad controller come up with your original normal trajectory, you still expect maybe, right? You'd expect your state and action at time T to be maybe reasonably similar to what even the sloppy controller had done, right? So you want a fly trajectory maybe you want to make a 90-degree turn. Maybe if a bad controller that does a pretty sloppy job, but at any given in time you're still moving around this trajectory. So this is really telling you where along, say, the 90-degree turn trajectory, just very roughly, where along the trajectory you expect to be at any given time and so let's linearize around that point. Okay?

Then you would – having found the linear model you run LQR to get the optimal policy for this specific linear model and now you have a better policy. And the final thing you do is – boy, I'll write this on a different board, I guess. Okay. Shoot. The last step is you use a simulator, a model, to come up with a new normal trajectory. So i.e., okay? So now you take the controller you just learned and basically try flying your helicopter in your simulator. So you initialize the simulator to the initial state, and I'll call the S bar zero, and you'll run every time step. You choose an action which I'll call A bar T, using the controller pi T that you just learned using LQR. And then you simulate forward in time, right? You use the simulator, the function F, to tell you what the next state S bar T plus one will be when your previous state and action is bar T A bar T. And then you linearize around this new trajectory and repeat. Okay? So now you have a new normal trajectory and you linearize your simulator around this new trajectory and then you repeat the whole procedure. I guess going back to step two of the algorithm. And this turns out to be a surprisingly effective procedure. So the cartoon of what this algorithm may do is as follows. Let's say you want to make a 90-degree turn on the helicopter let's see one, you know, a helicopter to follow a trajectory like that. Follow up of a very bad controller, I just, you know, hack up some controller, whatever. Have some way to come up with an initial normal trajectory. Maybe your initial controller overshoots the turn, takes the turn wide, right? But now you can use these points to linearize the simulator. So linearize in a very non-linear simulator and the idea is that maybe this state isn't such a bad approximation. That maybe a linearization approximation at this sequence of states will actually be reasonable because your helicopter won't exactly be on the states, but will be close to the sequence of states of every time step. So after one duration of DDP, that's the target trajectory, maybe you get a little bit closer and now you have an even better place

around to linearize. Then after another linearization of DDP you get closer and closer to finding exactly the trajectory you want. Okay?

So turns out DDP is a sort of – it turns out to be a form of a local search algorithm in which you – on each iteration you find a slightly better place to linearize. So you end up with a slightly better control and you repeat. And we actually do this – this is actually one of the things we do on the helicopter. And this works very well on many – this works surprisingly well – this works very well on many problems. Cool. I think – I was actually going to show some helicopter videos, but in the interest of time, let me just defer that to the next lecture. I'll show you a bunch of cool helicopter things in the next lecture, but let me just check if there are questions about this before I move on. Yeah?

Student:[Inaudible]

Instructor (Andrew Ng):In this sample? Yes, yeah, right, yeah. So I'm going to – let's pick some kind of horizon T. So I'm going to run through my entire trajectory in my simulator, so I end up with a new nominal trajectory to linearize around, right? Okay? Yeah?

Student: So does this method give you, like, a Gaussian for performing, like, a certain action? Like you talked about, like, the 90-degree turn thing or something.

Instructor (Andrew Ng):Right.

Student: So is this from one, like, is this from one, like, one 90-degree turn or can you [inaudible]?

Instructor (Andrew Ng):Yeah. So it turns out what – so this is used clear – let's see. Go and think about this as if there's a specific trajectory that you want to follow. I'm just gonna, car or helicopter or it could be in a chemical plant, right? If there's some specific sequence of states you expect the system to go through over time, so that you actually want to linearize at different times – excuse me. So, therefore, the different times you want different linear approximations, your dynamics, right? So I actually start to laugh over stationary simulator, right? I mean, this function F, it may be the same function F for all time steps, but the point of DDP is that I may want to use different linearizations for different time steps. So a lot of the inner loop of the algorithm is just coming up with better and better places around to linearize. Where at different times I'll linearize around different points. Does that make sense? Cool. Okay, cool. So that was DDP.

And I'll show examples of DDP results in the next lecture. So the last thing I wanted to do was talk about Kalman filters and LQG control, linear-quadratic Gaussian control. And what I want to do is actually talk about a different type of MDP problem where we don't get to observe the state explicitly, right? So far in every one I've been talking about, I've been assuming that every time step you know what the state of the system is, so you can compute a policy to some function of the state is in. If you've all ready had that, you

know, the action we take is LT times ST, right? So to compute the action you need to know what the state is.

What I want to do now is talk about the different type of problem where you don't get to observe the state explicitly. The fact – before we even talk about the control let me just talk about the different problem where – just forget about control for now and just look at some dynamical systems where you may not get to observe the state explicitly and then only later we'll tie this back to controlling some systems. Okay? As a concrete example, let's say as, sort of, just an example to think about, imagine using a radar to track the helicopter, right? So we may model a helicopter, and this will be an amazingly simplified model of a helicopter, as, you know, some linear dynamical systems. So [inaudible] ST plus one equals AST plus WT, and we'll forget about controls for now, okay? We'll fill the controls back in. And just with this example, I'm gonna use an extremely simplified state, right? Where my state is just a position in velocity in the X and Y directions, so you may choose an A matrix like this as a – okay?

As an extremely simple – as a, sort of, an extremely simplified model of what the dynamics of, like, a plane or object moving in 2-D may look like. So just imagine that you have simulation and you have a radar and you're tracking blips on your radar and you want to estimate the position, or the state, of the helicopter as just as its XY position and its XY velocity and you have a very simple dynamical model of what the helicopter may do. So this matrix, this just says that XT plus one equals XT plus X star T plus noise, so that's this first equation. The second equation says that X star T plus one equals 0.9 times X star T plus nine. Yes, this is an amazingly simplified model of what a flying vehicle may look like.

Here's the more interesting part, which is that with – if you're tracking a helicopter with some sensor you won't get to observe the full state explicitly. But just for this cartoon example, let's say that we get to observe YT, which is CST plus VT where the VT is a random variables – Gaussian random variables with, say, zero mean and a Gaussian noise with covariance given by sigma V. Okay? So in this example let's say that C is that and – so CST is equal to XY, right? Take this state vector and multiply it by Z, you just get XY. So let's see what the sensor, maybe a radar, maybe a vision system, I don't know, something that only gets to observe the position of the helicopter that you're trying to track.

So here's the cartoon. So a helicopter may fly through some sequence of states, let's say it flies through some smooth trajectory, whatever. It makes a slow turn. So the true state is four-dimensional, but I'm just drawing two dimensions, right? So maybe you have a camera sensor down here, or a radar or whatever, and for this cartoon example, let's say the noise in your observations is larger in the vertical axis than the horizontal axis. So what you get is actually one sample from the sequence of five Gaussians. So you may observe the helicopter there at times step one, observe it there at time step two, observe it there at time three, time four, time five. Okay? All right. So that's what your – there's a sequence of positions that your camera estimate gives you. And given these sorts of

observations, can you estimate the actual state of the system? Okay? So these orange things, I guess, right? Okay?

These orange things are your observations YT. And test for the state of helicopter every time. Just for it, so the position of the helicopter at every time. Clearly you don't want to just rely on the orange crosses because that's too noisy and they also don't give you velocities, right? So you only observe the subset of the state of variables. So what can you do? So concretely – well, you don't actually ever get to observe the true positions, right? All you get to do is observe those orange crosses. I guess I should erase the ellipses if I can. Right. You get the idea. The question is given – yeah. You know what I'm trying to do. Given just the orange crosses can you get a good estimate of the state of the system at every time step?

So it turns out that – well, so what you want to do is to estimate the distribution on the state given all the previous observations, right? So given observations, you know, one, two, three, four, and five, where is the helicopter currently? So it turns out that the random variables, S zero, S one, up to ST and Y1 are to ST, have a joint Gaussian distribution, right? So one thing you could do is construct a joint Gaussian distribution – can define vector value random variable Z, S zero, S one, up to ST, Y1 up to YT, right? So it turns out that Z will have some Gaussian distribution with some mean and some covariance matrix. Using the Gaussian marginalization and conditioning formulas. But I think way back when we talked about factor analysis in this class, we talked about how to compute marginal distributions and conditional distributions of Gaussians. But using those formulas you can actually compute this thing. You can compute, right? You can compute that conditional distribution. This will give a good estimate of the current state ST. Okay?

But clearly this is a extremely computationally inefficient way to do so because these means and covariance matrixes will grow linearly with the number of time steps as you're tracking a helicopter over tens of thousands of time steps. They were huge covariance matrixes, so this is a conceptually correct way, but just a computational not reasonable way to perform this computation. So, instead, there's an algorithm called the Kalman filter that allows you to organize your computations efficiently and do this. Just on the side, if you remember Dan's discussion section on HMM's the Kalman filter model turns out to actually be a hidden Markov model. These Kalman's are only for those of you that attended Dan's discussion section. If not then what I'm about to say may not make sense. But if you remember Dan's kind of section of the hidden Markov model, it actually turns out that the Kalman filter model, this linear dynamical system with observations is actually an HMM problem where – let's see. Unfortunately, the notation's a bit different because Dan was drawing from, sort of, a clash of multiple research communities using these same ideas. So the notation that Dan used, I think, was developed in a different community that clashes a bit with the reinforcement learning community notations. So in Dan's notation in the HMM section, Z and X were used to denote the state and the observations. Today, I'm using S and X to denote the state and the observations. Okay?

But it turns out what I'm about to do turns out to be a hidden Markov model with continuous states rather than discrete states, which is under the discretion section. Okay. If you didn't attend that discussion section then forget everything I just said in the last minute. So here's the outline of the Kalman filter. It turns out that, so it's a cursive algorithm. So it turns out that if I have computed P of ST given Y1 up to YT, the Kalman filter organizes these computations into steps. The first step is called the predict step. Where given P of ST – where you all ready have P of ST given Y1 up to YT and you compute what P of ST plus one given Y1 up to YT is. And then the other step is called the update step. Where given this second line you compute this third line. Okay? Where having taken account only observations of the time T you know incorporate the lots of the observations up to time T plus one.

So concretely – oh, and let's see. In the predict step it turns out that – so what I'm going to do is actually just outline the main steps of the Kalman filter. I won't actually derive the algorithm and prove it's correct. It turns out that, I don't know, working out the actual proof of what I'm about to derive is probably significantly – it's probably, I don't know, about as hard, or maybe slightly easier, than many of the homework's you've done all ready. So and you've done some pretty amazingly hard homework, so you can work out the proof for yourself. It's just write out the main outlines and the conclusion of the algorithm. So for the acceptance of the vest T given Y1 after YT. If that is given by that then where – okay? So given ST, having computed the distribution ST given Y1 through YT – and computed the distribution of ST plus one given Y1 through YT as Gaussian, with this mean and this covariance, where you compute the mean and covariance using these two formulas. And just as a point of notation, right? I'm using ST and YT to denote the true states in observations. So the ST is the unknown true state. Okay? ST is whatever state this one is in and you actually don't know what ST is because you don't get to observe this.

And, in contrast, I'm using these things like ST given T, ST plus one given T, sigma T given T, and so on. These things are the results of your computations, right? So these things are actually things you compute. So I hope the notations are okay, but these – ST is the unknown true state, right? Whereas these things, ST equals one given T and so on, these are things that you compute inside your algorithm. Okay? So that was the predict step.

And in the update step, you find that – well, okay? And so that's the updates of the Kalman filter where you compute this in terms of your ST given Y1 through YT. So after having performed the most recent Kalman filter update you find that, right? Your perceived distribution on the estimate of ST plus one, given all your observations so far, is that it's Gaussian with mean given by this and variance given by that. So, informally, this thing ST plus one given T plus one is our best estimate for ST plus one, right? Given all the observations we've had up to that time. Okay?

And, again, the correctness of these equations – the fact that I'm actually computing this mean and covariance of this conditional Gaussian distribution, you can – I'll leave you to sort of prove that at home if you want. Okay? I'm actually gonna put this together with

LQR control in a second, but, so before I do that let me check if you've got questions about this? Actually let me erase the board while you take a look at that. Right. Okay. Any questions for Kalman filters? Yeah?

Student: How is it computationally less intensive than compute some drawing Gaussian distribution and then find the conditional –

Instructor (Andrew Ng):Very quickly. Yeah, of course. So how is this less computationally intensive than the original method I talked about, right? So in the original method I talked about – wow, this is really back and forth. I said, let's construct a Z, which was this huge Gaussian thing, right? And figure out what the mean and covariance matrix of Z is. So sigma will be like R – it'll be – well, it'll be roughly, right? A T by T matrix, right? This is actually – or the T by T is actually T times number of state variables plus number of observation variables by that. This is a huge matrix and as the number of times it increases sigma will become bigger and bigger. So the conditional and marginalization operations require things like computing the inverse of T or subsets of T. So the naïve way of doing this will cos on the order of TQ computation, if you do things naively, right? If – because inverting like a T by T matrix cos on the order of TQ, roughly.

In contrast, the Kalman filter algorithm, like I said, over here. I just have the update step. On the other board I had the predict step. But you can carry out the computation on both of these lines and it's actually constant time. So on every time step you perform these Kalman filter updates. So if every time you get one more observation you perform one more Kalman filter update and the computation of that doesn't depend on it's – or the one time for every time step. So the amount of stuff you need to keep around in memory doesn't grow linearly with the number of time steps you see. Okay? Because – actually what – I think I just realized why – so, yes. Actually this is the way we actually run Kalman filters, which is initially I have just my first observation. So I then compute P of X1 given Y1, right? And now I know why I think my helicopter is at time step one. Having computed this there may be some time passes, like a second passes, and then I get another observation and what I'll do is I'll combine these two together to get P of X2 given Y1 and Y2, right? And then may be another second passes in time and I get another observation. So my helicopters move a little bit more, because another second's passed and I get another observation. What I do is I combine these two to compute P of SV given Y1, Y2, Y3. And it turns out that in order to compute this I don't need to remember any of these earlier observations. Okay? So this is how you actually run it in real time say. Okay? Cool. So – oh, drat, running out of time. The last thing I want to do is actually put these things together. So putting it together – putting Kalman filters together with LQR control you get an algorithm called LQG control, which stands for linear-quadratic Gaussian. But in this type of control problem, we have a linear dynamical system. So I'm now adding actions back in, right? So now B times AT. Okay? And then, so LQG problem, or linear-quadratic Gaussian problem, I have a linear dynamical system that I want to control and I don't get to observe the states directly. I only get to observe these variables YT. Okay? So I only get noisy observations of the actual state. So it turns out that you can solve an LGG control problem as follows. At every time step, we'll use a

Kalman filter to estimate the state, right? So concretely – let's say you know the initial state. Then you initialize this to be like that. If you know that the initial state is some state as zero, you initialize that as zero and that or, whatever, right? And this is just – well, okay? If you don't know the initial state exactly, then this is just a mean of your initial state estimate and that would be your covariance or your initial state estimate. So just initialize your Kalman filter this way. And then you use the Kalman filter on every step to estimate what the state is. So here's the predict step, right? Previously we had ST plus one give T equals – and so on. So this is your predict step and then you have an update step, same as before. The one change I'm gonna make to the predict step is now I'm going to take this into account as well. This is just saying suppose my previous state was ST given T, what do I think my next state ST plus one given T will be given no other observations and the answer is, you know, it's really just this equation, AST given T plus BAT.

And then, so this takes care of, sort of, the observations. And then the other thing you do is compute LT's using LQR, right? Assuming – then the other thing you do is you just look at the linear dynamical systems, and forget about the observations for now, and compute the optimal policy – oh, right. Previously we had that you would choose actions AT equals to LT times ST, right? So the optimal policy we said was these matrixes, LT times ST. So the other part of this problem you would use LQR to compute these matrixes LT, ignoring the fact that you don't actually observe the state. And the very final step of LQR control is that – well, when you're actually flying a helicopter, when you're actually doing whatever you're doing, you can't actually plug in the actual state because in LGG problem you don't get to observe the state exactly. So what you do when you actually execute the policy is you choose the action according to your best estimate of the state. Okay?

So in other words, you don't know what ST is, but your best estimate of the state at any time is this S of T given T. So you just plug those in and take LT times your best estimate of the state and then you go ahead and execute the action AT on your system, on your helicopter, or whatever. Okay? And it turns out that for this specific class of problems, this is actually optimal procedure. This will actually cause you to act optimally in your LQG problem. And there's this intuition that, earlier I said, in LQR problems it's almost as if the noise doesn't matter and in a pure LQR problem the WT terms don't matter. It's as if you can ignore the noise. So it turns out that by elaborating that proof, which I'm not gonna do you can – you're welcome to proof for yourself at home. It's that intuition means that you can actually ignore the noise in your observations as well. The ST given T is some of your best estimate. So it's as if your true state ST is equal to ST given T plus noise. So in LQG control, what we're going to do is ignore the noise and just plug in this ST given T and this turns out the optimal thing to do. I should say, this turns out to be a very special case of a problem where you can ignore the noise and still act optimally and this property – this actually is something called the separation principle where you can design an algorithm for estimate the states and design an algorithm for controlling your system. So just glom the two together and that turns out to be optimal. This is a very unusual property and it pretty much was true only for LQG. It doesn't hold true for many systems. Once you change anything, one's that's non-linear, you know, some other noise

model of one that's non-linear once this – I don't know. Once you change almost anything in this problem this will no longer hold true. The – and just estimate the states and plug that into a controller that was designed, assuming you could observe the states fully. But that once you change almost anything this will no longer turn out to be optimal. But for the LQG problem specifically, it's kind of convenient that you can do this. Just one quick question to actually close

Student:[Inaudible]

Instructor (Andrew Ng):Oh, yes. Yeah. In every embassy wing – in everything I've described I'm assuming that you're already learned A and B or something, so to –

Student:[Inaudible]

Instructor (Andrew Ng):Yeah, right. Okay. Sorry we're running a little bit late; let's close for today and next time I'll talk a bit more about these partially observed problems.

[End of Audio]

Duration: 79 minutes