MITOCW | Lecture 10 | MIT 6.832 Underactuated Robotics, Spring 2009

RUSS TEDRAKE: OK, welcome back. So last week, we spent the week talking about policy search methods, and trying to make a distinction between those and the value-based methods we started with. And by the end of the week, we had a couple pretty slick methods for optimizing an open-loop trajectory of the system. So we talked about at least two ways.

So by open-loop, I mean it's a function of time, not a function of state. We talked about the shooting methods, where we evaluated J of alpha x0 times 0 just by simulation. And we evaluated-- explicitly evaluated the gradients by-- well, I gave you two algorithms for it.

I gave you one that I called back prop through time-- which was an adjoint method-- and another one that I called RTRL-- real-time recurrent learning, which are the names from the neural network community, but perfectly good names for those methods.

And then the claim was that, if you can compute those two things by simulation or-- forward simulation and then a back propagation pass, or a simulation, which carried also the derivatives forward in time, then we could hand those gradients to SNOPT or some other non-linear optimization package.

And if we're good, we can also lean on SNOPT to handle things like final value constraints. If you want to make sure the trajectory succeeds in getting you exactly to the goal or if you want to make sure that your torques are never bigger than some maximum talk allowed, then you can take advantage of that.

And the second method, remember, was direct co-location method, which we often abbreviate as DIRCOL. And the big idea there was to over-parameterized our optimization with the open-loop trajectory, but also the state trajectory, which makes coming up with gradients simple.

And then I have to enforce the constraint that x of-- let's say, in discrete time here, n plus 1 had better be subject to the dynamics-- so two very similar methods of trying to compute some open-loop trajectory as a function of time. Ultimately, what I care about is a set of actions that I apply over time that will get me to the goal or minimize my cost function.

In the case where I explicitly parameterized an open-loop trajectory, both of these results in a solution which satisfies the Pontryagin minimum principle, subject to discretization errors and things like that.

AUDIENCE: [INAUDIBLE]

RUSS TEDRAKE: We did, right. So I should say, subject to time discretization. That's the one place where technically, it would satisfy a discrete time version of Pontryagin's minimum principle.

AUDIENCE: [INAUDIBLE]

RUSS TEDRAKE: You can think of it whichever way it makes you happier-- so in fact, the parameters that you hand in-- maybe it's easier to think of it as a function-- a discrete function of time, because you're going to hand it u at certain points in time, and you're going to handle x-- hand it x at certain points in time. And this discrete time update can be an Euler integration or a higher order integration of your continuous dynamics, but you only satisfy the constraints of discrete intervals of time. Yeah.

OK, I did give you a slightly more general-- I tried to point out that these methods could equally well compute, find good parameters of a feedback control or something too. The simple case was when my parameters alpha were explicitly my control tape, but more generally, if you wanted to tune a feedback controller-- a linear feedback controller, or a non-linear feedback controller, or a neural network, or whatever it is, you can use the same methods to do that.

I would only make this statement in the case where the controller specifically is the open-loop tape, because if I parameterized my trajectory by some feedback controller, for instance, then that's going to restrict the policy class. That's going to restrict the class of tapes that I can look over, which makes it a more compact, more efficient way to solve your optimization, but potentially prevents you from achieving a perfect minimum.

AUDIENCE: [INAUDIBLE]

RUSS TEDRAKE: Yep. So by virtue of saying that they satisfy Pontryagin's minimum principle, we know that that's only a local optima. This says that I can't make a small change in u to get better performance. Yep. But it's only a necessary condition for optimality, not a sufficient one. But there's a bigger problem with it-- with both of those.

And that's the fact that they're completely useless in real life, unless I do one more step, which is to stabilize the trajectory as I get out. So finding some open-loop trajectory by these methods, satisfying Pontryagin's minimum principle-- fine. But there's nothing in this process that says, if I don't-- if I changed my initial conditions by epsilon, I could completely diverge when I follow-- when I execute that open-loop trajectory.

If I change my simulation time step by a little bit, I might diverge. If I have modeling errors, I might diverge. So in order to make these useful for a real system, we have to do another step, which is trajectory stabilization. And it actually follows quite naturally from the things we've already talked about. OK, so today we're going to give these guys teeth with a trajectory optimization.

And I'll show you examples of a trajectory that's optimized beautifully for the pendulum even, and if I simulate it a little differently back-- just does the wrong thing. It never gets to the top. So we want to get rid of that problem. OK, so the solution is to design a trajectory stabilization.

Now, for those of you that have been playing with robots for many years, when you hear trajectory stabilization, what do you think of? What kind of tricks to people use for trajectory stabilization?

AUDIENCE: Sliding surfaces.

RUSS TEDRAKE: Sliding surfaces-- that's a good one for-- [INAUDIBLE] often will design a sliding surface and squish the aerodynamics. That's actually pretty encompassing. I think a lot of the trajectory stabilizers are based on sliding modes or feedback linearization in some form.

And all I'll say about it is that the story's sort of the same as everything we've said. If you have a fully actuated system, it's not hard to design a trajectory stabilizer. A good sliding mode controller could take-- could work even for an underactuated system, but I think there's a-- I prefer the linear quadratic form of these trajectories stabilizers.

OK, so we want to do a trajectory stabilization that's suitable for underactuated systems. And the approach is going to be with LQR. OK, so if we're going to use LQR, we better be able to linearize our system. So far, when we've done the linearizations, we've only done them at fixed points.

So the first thing we have to ask ourselves is, what happens if we try to linearize at a more arbitrary point in state space? Yeah. So let's say I've got the system x dot equals f of xu, and now I want to linearize around some x0, u0, but not necessarily a carefully chosen x0, u0-- just something random in state space.

The Taylor expansion of this says that this thing's going to be approximately f of x0, u0, partial f, partial x evaluated at x. OK, and we called this before A and this B, and so that thing we can actually write as, in general, in the case where f of x0, u0-- if x0, u0 was a fixed point of the system, that term disappears, but be careful. If you're doing your linearization out here, if you're at-- not at a fixed point, if you have any velocity, for instance, then, in the original x-coordinates, it's not actually-- the Taylor expansion doesn't give you a linear system. It gives you some affine system. This thing is harder to work with-- not incredibly harder, but harder to work with.

The solution is quite simple, but I just wanted to say it the bad way first so that you appreciate the good way. If we change coordinates and we use instead for our coordinates the difference between x and x0 of t, then x bar dot is going to be x dot minus x0 dot equals x dot minus f of x0, u0, which is that C.

This guy here is taken care of in this new coordinate system, which allows me to write the whole thing as x bar dot equals A of x bar. You with me on that? Linearizing a system at a more arbitrary point-- doing a Taylor expansion results in a linear system only if you change coordinates to lie on some system trajectory.

So x0, u0 must be a solution of x of f of xu of that equation. And then the system reduces to a linear system description. But the cost you pay for this beautiful, simple-- well, let me be even a little bit more careful. So A here, this partial f, partial x, is evaluated at xt u of t. And in general, A and B in this-- when I do this are functions of time, as well as x and t.

That's a pretty important point. So if I'm willing to change coordinates to live along the trajectory, then the result is I can get this linear time-varying model of the dynamics along feasible trajectories-- system trajectories. The cost is that you have to work in a coordinate system that moves along your trajectory. So we'll see where that comes in in a little bit.

But the first question is, OK, let's say I've got this linear time-varying-- time-varying linear system. Can I do all the things I want to do with that? In most of our control classes, we end up doing LTI systems. LTV systems-- linear time-varying-- are actually a fantastically rich class of systems that we don't talk about enough, I think, in life.

They're still linear systems. Superposition still holds. If I have initial condition 1 and some u trajectory 1 for t greater than equal to t0, and that gives me some resulting x trajectory out for t greater than t0, and I have another solution with a different initial condition and a different control, and that gives me a different-- I call this x1, x2 for t greater than or equal to t0-- if I have that, then it better be the case that alpha 1 x1 of t0 plus alpha 2 x2 of t0 plus alpha 1 u1 tape plus alpha 2 u2 tape is going to result in a trajectory which is alpha 1 x1 plus alpha 2 x2.

That's superposition. That's the defining characteristic of linearity. And even though this is a richer class of systems-- these A of t, x of t, B of t, u of t-- superposition still holds. And in fact, a lot of our derivations that we've done that are for linear systems still hold. OK, so now the question is, how do we design-- how do we work with the fact that this thing is still easy, and design a controller that works with this new linearized system?

Maybe first I should break out my colored chalk and make sure we have intuition about this. Do you understand what this is doing, if I do this time-varying linearization? Let me do an example with the pendulum here, our favorite theta, theta dot. And let's say we carve up-- we find some nice solution which gets me from my one fixed point to the other fixed point.

The ones we were getting were these pump-up trajectories, which looked something like this. I'm moving through state space here, and the dynamics here vary with state in a non-linear way. But if I have a trajectory, a feasible trajectory that goes through the relevant parts of state space, then this time-varying linearization takes my non-linear system, and makes it parameterized only-- instead of by being parameterized by state, it's going to make it parameterized only by time along the trajectory.

The trick is the trajectory allows me to reparameterize my non-linearity in terms of time, instead of state. It sounds like a simple thing-- I'm just reparameterizing it-- but it makes all the difference in the world. If things are parameterized as a function of time, and are otherwise linear, then I could do all kinds of computation on them. I can integrate the equations. I can design quadratic regulators on it. It makes all the difference in the world.

So what I'm effectively doing is coming up with local linear representations of the dynamics along the trajectory. I'm not sure if this is a helpful way for me to draw it, but you can think of this thing as approximating the dynamics along that trajectory. At every given instant in time, I'm going to use one of these linear models.

This is supposed to be some plane that you're driving through-- not sure if that's actually helpful graphic, but it's the way I think of it. And by virtue of taking a particular path through, I can make locally linear models on which these things have eigenvectors, and eigenvalues, or whatever that are valid in the neighborhood of the trajectory.

So if you can imagine, even without any stabilization, it could be that I could quickly assess the stability of my time-varying linear model. And trajectories in this linear model may converge to the nominal limit cycle, or they may diverge, depending on A and B. Or they may blow up.

This is by far the more common case, unfortunately. You'd be very lucky to come out of a shooting method or a direct co-location method, and end up with a system where if you played it out, it just happened to be a stable trajectory. But we can assess all that quickly with these time-varying linearizations found locally. Make sense? Yeah?

AUDIENCE: [INAUDIBLE] talk about that there is a bad way of doing this. This is not a bad way of doing this, right? We were talkinga about it.

RUSS TEDRAKE: If I do a Taylor expansion of my system in the original coordinate system, which is x, then it's not linear. End parentheses, that was the bad way to do it. Yeah? Good way to do it-- change the coordinates to a coordinate system, which moves with the trajectory. If you do that, things become time-varying linear. That was a good way to do it, and that's still in open parentheses. We're still going. Yeah.

OK, so our task now is to design a time-varying feedback controller-- since our model is time-varying, you'd expect our solution to also be time-varying-- which takes these bad, unstable trajectories of the system-- and they really are-- I'll show you simple pendulum. This trajectory comes out.

Actually, if you just integrate in a different way, it'll go off and do the wrong thing. It typically doesn't go off and add energy to the system so much. The ones I get-- I see, I'll show you, are more-- they diverge and the other way, and end up just floating around here, for instance. But they're not going to get you up here. So can we design a time-varying stabilizer that regulates that trajectory?

OK, I did actually do the original finite horizon LQR derivation on the board that day-- definitely won't write all that again, but let me say that roughly nothing in that derivation breaks-- I'm going to show you the important pieces-- nothing in that derivation breaks, surprisingly, if A and B are now a function of time.

So let's remember that-- the LQR derivation. Now I'm working with this x bar coordinate system. And I want to design a cost function to minimize here, which lives in this coordinate system again here. Let's say it's the final horizon times Qf-- I've been trying to use t little f, since my transposes look like the final horizon time otherwise-- 0 to tf dt x bar-- again, transpose Q plus u bar Ru.

OK, in the original LQR derivation, we guessed that the form-- that the optimal policy had the form x bar S of t x bar. That's still intact. That's still a good assumption. This thing's linear. It's just in a different coordinate system.

And we started cranking through the sufficiency theorem, the Hamilton-Jacobi-Bellman equation. And we found that our optimal feedback policy-- first of all, our optimal cost-to-go was described by this Riccati equation, which was negative S of t is Q minus S of t B our inverse B transpose S of t plus S of t A plus A transpose S of t.

And it turns out that, with the-- if you have a time-varying A and B, that it's-- exact same dynamics govern it. You just have your time dependence also in A and B. And that exact same Riccati equation works, and our final value condition was just Qf.

And you can see from this, if it didn't make a difference for me when A and B became functions of time, it's pretty simple-- although less interesting, I guess. If Q were to be a function of time-- no problem. If R was a function of time-- no problem. They still have to be positive definite and symmetric. Oops-- I did it the wrong way. Q can be 0, but R can not be 0.

OK, so the LQR you know and love, that you've used in Matlab, is the time invariant infinite horizon LQR. I told you that, if you cared about a finite horizon and you had a time invariant linear system, then suddenly you had to-- you couldn't just find the stationary points in this.

Remember, Matlab's solution just tells you the long-term behavior of S. In the time-- finite horizon time, even the LTI case, which is the A and B do not depend on time-- the linear time invariant case-- I still had to integrate back this Riccati equation in order to get my LQR controller. It's no more expensive to do the same thing in the linear time-varying feedback case.

And the resulting controller is-- u star is my nominal controller minus my R inverse B transpose S of t x bar. These equations come up enough that these are pretty famous, pretty important equations, and so I-- those I know off the top of my head. They come up all the time. And this is the resulting optimal trajectory, which is my nominal trajectory plus my feedback gain, which came out of my original LQR controller, if you remember that.

AUDIENCE: [INAUDIBLE]

RUSS TEDRAKE: Yes-- good. I should definitely put a T under B. Thank you. I haven't written that case, but R could equally well be time-dependent. OK, so something big just happened. I can take a really, really complicated non-linear system along some trajectory-- if I find a good trajectory, then I can actually linearize that system along this trajectory and stabilize it.

The thing I haven't convinced you of yet-- because I only know how to do it from showing examples, but it really works well. So even though it's a linear system-- it's a linear approximation of the non-linear system, something like the [INAUDIBLE] or the cartpole swing-up. It's got a huge basin of attraction. Lots and lots of initial conditions will find their way to the trajectory and get to the goal.

If you want to do non-linear control of a humanoid robot or something like this, this actually scales pretty nicely. I just have to solve this equation. S is the size-- is a matrix that's by number of states. But I could do that in 30 dimensions. That's no problem. And even for very non-linear systems, local linear feedback works very, very well-- so well, in fact, that I think that, if you ask-- and when I did ask the [INAUDIBLE] guys, Sasha Megretski says, this is definitely what I would do if I was controlling a walking robot or something like that.

We're trying to do the same thing to control neurons in a dish now. We're trying to build good models of the dynamics-- time-varying models, for instance-- and then doing this kind of control. Yeah. It works really, really well. The only complaint about it is that it's going to have-- it's based on this linear approximation, so it will have a finite basin of attraction.

For some systems, it can be quite big. If you have systems with hard non-linearities, it won't be as big. Later in the course, I'll show you ways to explicitly reason about the size of those basins of attraction, but today let's just say this is a good thing to know, good thing to have in your pocket. Let me show you a working-- try to convince you that it's pretty good.

OK, so let's see where I've left myself here. I took this-- the pendulum-- let's do the shooting version. They both work fine, but let's do the shooting version. Is that bigger than I did last time? That's pretty obnoxious. Maybe it's always been obnoxious. Can we get away with that? Yeah. You guys are like, I'm not blind.

OK. So I showed you last time the shooting code. It comes out with a resulting tape x, t, and u. After the result of these trajectory optimizers, whether it's shooting or whether it's direct co-location-- whatever it is-- it comes up with some open-loop tape. I put x in there too just to-- as the reference trajectory that results, but what really matters is the time stamps and u command, the open-loop tape.

Why don't I save it this time? OK, so it comes up-- in this case, with these parameters I've chosen, comes up with some one-pump policy. With the torque limits I have, the [INAUDIBLE] I have, it comes up with a one-pump policy that gets me to the top in four seconds. OK, let me now just simulate that a little bit differently.

So the only thing I'm going to do here now is-- this control_ode is just a simulation which plays back exactly the same open-loop tape, but it plays it back with a little more careful integration-- because in the actual-- in the shooting code I used, I used the big time step just so I don't waste time computing gradients to the n-th degree of accuracy. That's not worthwhile.

If I simulate the exact same thing back with a more careful ode integration, let's see what happens. So that was that same trajectory that-- exact same control inputs, just simulated more carefully. It made its honest effort to get up there, but it didn't quite get up there, turned around, and came back down.

I'm trying to show it also in just-- this is the different state trajectories over time. You can see that the red and blue lines are the desired versus actual in the-- in theta, in this case. And these two lines are the desired versus actual in theta dot. They start off exactly on top of each other, but just little differences in the numerics causes them to go in different directions-- part ways.

OK, so now I've got this LTV LQR solution, which is exactly what I just showed you. So I was just simulating a just now with just u being the nominal u. Now I'm going to add this time-varying feedback term, x minus x desired. And now my more careful integration results in a closed-loop system, which not only got to the goal, but actually stayed up at the goal, because I have a stable system all the way to the top. OK?

All right, so what I just said was very unimpressive. I said I computed a open-loop policy with my methods from Thursday. I simulated them back. They didn't work. But I then put a feedback controller on, and from the exact same initial conditions, I now can simulate them, and they work. So it's disappointing that we had to do that at all, but I can now-- the stability is more than just stabilizing the initial conditions. Let's add some fairly big random numbers to that initial condition and see what happens.

It's recomputing the policy every time, just because it was fast enough that I didn't bother to change it. OK, so that actually started with pretty big different initial conditions. So theta was off by-- I don't know-- 2/10 of a radian or something like this. The velocities were off by 1/2 a radian per second. We could crank that up. I bet it does a lot better than that.

But if you watch these things, they converge quite nicely to together at the end there. And what matters is they get up to the top. So again, these things come together, find their way up to the top, and live. I bet, if I put it a lot bigger, it'll still work. I normally do an order of magnitude, but let's not be silly.

Oh-- didn't make it. There's only one reason it didn't make it, actually. It's because, if you look in here, I'm actually honest about implementing the max torques. Yeah. So I actually have a torque limit, I impose it, and it lives on there. If I didn't, I bet I could convince you it works for any initial condition. But let's try it one more time-- get a little more lucky with the initial conditions.

Oh, come on. Come on. Yes. OK, that was pretty far off, and it's still found its way back to the trajectory. Good-- yeah? Look at how big those initial conditions are. There and there versus-- wow, that's really good. OK. Did I see a question? No? All right, so this stuff works for pendulum. It works for more interesting systems too. I'll just show you the cartpole real quick here.

I won't do the-- here is what it looks like without feedback. I'll just do the initial conditions corrupted solution, pump up-- OK, so if you remember my solutions from last time, I never drove off the screen before, so that it was actually it catching it by deviating enough that it came off the screen, and then slowly coming back to the top.

It must be its x position or something going way off. No, not x position-- what is that? That's my control. Yeah. Did I do torque limits on that one? I still did torque limits. I just set them high, I guess. Yeah. So it really works. And the cool thing is the cost of implementing that LQR LTV stabilizer was negligibly more than implementing the-- most of that time was the shooting optimization. Yes?

AUDIENCE: Why do you always start at the 0 time? You could look at the initial conditions and look where is the closest point on my nominal trajectories and then do your control policy from that moment in time.

RUSS TEDRAKE: OK. So that's a really, really good. OK, that's exactly what I want to talk about next, actually. I designed a time-varying feedback controller, is negative K of t x bar of t. I designed that ahead of time. And then, from the initial conditions, I started simulating from 0, and I just played out the-- my nominal trajectory just marched forward with time, my feedback controller just marched forward with time, and my aerodynamics just marched forward with time.

OK, so before I explicitly address your question, let me point out-- let me ask even a simpler question here. If I had plotted that in state space, what you would have seen is that the trajectory starts off somewhere in state space and comes together. That would have a good idea. Maybe I should do that in a minute. But it comes together and finds its way onto that trajectory. Yeah?

OK, so here's the question. Instead of just changes in initial conditions, what happens if I have disturbances that push me off the trajectory? Well, that's OK. That's no different really than a different initial condition. They'll come back on here.

What happens if I have a disturbance that pushes me along the trajectory with this controller? Let's say I've got the helpful disturbance, which, when I was right here, just happened to push me right to there. What's my feedback controller going to do?

AUDIENCE: Slow it down.

RUSS TEDRAKE: Yeah-- probably in a dramatic fashion. It's the same way-- it tries to quickly converge from here. It's going to push itself back towards that point, possibly. Slowing down doesn't-- makes it sound-- no big deal. It can't go backwards, but it might try to do something more severe to try to catch up with that old trajectory.

So the major limitation of this is that it's blindly-- in order to have the strong convergence properties that we have, the controller is blindly marching forward in time. The great thing about switching to a time parameterization I can compute everything-- everything's linear again. The bad thing is you're a slave to time.

So Phillip asked a next question. He says, so why not-- why do I just blindly start marching forward from time 0? Maybe, if I have a controller, I should just look for the closest point in my trajectory, and then, instead of indexing off time, index off some sort of phase, some fraction of my trajectory, and then execute that controller.

And you can do that. I wish you the best if you do that, but my suspicion is that, if on every dt, you pick the closest point in the trajectory, then the result is you're going to chatter like you wouldn't believe. So there's a lot of protection you get when you-- you could think of this very much as a gain-scheduled linear controller.

This is a time-varying gain scheduling, and the problem is if I switch gain quickly, then you're going to get chattering. So it might make a lot of sense, for instance, if you were to get a big disturbance, to re-evaluate, and try to find the closest point, and start executing that new policy with time re-indexed. But it's probably a bad idea, in my experience, to decide which part of the trajectory you're closest to on every-- every dt. That's probably a bad idea. Yes?

AUDIENCE: Could you maybe play some tricks if you had some idea of the basin of attraction of the current point you're trying to get to? And if you know that you're outside of it, then work around it, [INAUDIBLE]?

RUSS TEDRAKE: Yes. So I have a particular trick the does that does that in-- we'll talk about it in the motion planning, but-- yeah, so Mark knows about these tricks for computing basins of attraction pretty efficiently. And so these days what we do is we actually try to compute the funnel-- the basin of attraction of this trajectory around the trajectory, and you could know discretely if you left that basin of attraction.

So I'll give you the recipe for that, but it actually makes more sense, I think, in the motion planning context, where we actually will design trajectories that fill the space with these basins. This is very similar to the concept of flow tubes. Yes.

OK, so big idea-- turn my non-linear system into a linear time-varying system, because I've re-parameterized it that along the trajectory. Do linear time-varying control, and even really complicated systems-- it'll work well. We're doing on our [INAUDIBLE] plane. I mean, it's really a pretty good idea.

When I first started working with it, I thought that it would have the problem that-- it would have the property that it uses a lot of control to force itself back to the trajectory and rigidly follow the trajectory. It's easy to equate linear control with high-gain linear feedback, which people do a lot of, but it doesn't necessarily need that property.

If R is small in this derivation, it can actually take very subtle approaches back to the trajectory. Your system might come in and do whatever it needs to get back on the trajectory with very little torque. The only price you pay is, if your torque is smaller, if you're penalizing torque use higher, then you might restrict your-- that might shrink your basin of attraction. It might be that, because it's trying to use less torque, it will not overcome the non-linearities.

But in the neighborhood of the trajectory, you can get these very elegant solutions which look like minimal energy kind of solutions for the non-linear problem in the vicinity of these trajectories. So one of the ideas we'll talk about later is how do you design the minimal set of trajectories-- which, if you use these controllers, which do the right thing in a lot of places-- if you walked away from this class knowing nothing but direct co-location and linear time-varying feedback control, I bet you could control a lot of cool systems.

Yeah. I guess you also have to know sys id, which I'm not going to tell you about. That's the gotcha. You have to have a model for all this stuff. If someone gives you a model, if you're willing to construct a model, then you can do a lot of things with this.

OK, I want to give you one more mental picture to think about what this is doing so it launches into the next thing here. So my cost-to-go function, which I just erased, is, remember-- my cost-to-go function, J of x bar t, is x bar S of t x bar. This is a quadratic form. Just like the original LQR, you can think of this as a quadratic bowl.

In the LTI LQR case-- am I OK throwing around these three-letter acronyms? In the LTI LQR case, it was a static quadratic bowl centered around the point I'm trying to stabilize-- so my cost-to-go. It said-- says, as I move away from the point I'm trying to regulate, I'm going to incur more cost in the direction-- the rate it grows depends on the variables inside S.

Now, in this picture, I have still a time-varying-- I have a time-varying quadratic bowl, but it's also moving through time, because it's based on x bar. So in my pendulum world, if I have this nominal trajectory, you can think of it as having some quadratic bowl here. And the LTI stabilizer that we did come up with that was based on LQR did have some sort of quadratic bowl shape that looked like that.

Backwards in time, there's going to be another quadratic bowl. Can I draw it very badly like this? If I can just draw coming off the board a little bit-- so there's some quadratic bowl centered around this point, which is my costs-to-go. At that point, if I marched further backwards in time, I've got some other quadratic bowl around this point.

That makes the point, again, that-- if my quadratic ball is currently this because time is 5-- or I had a 4-second trajectory-- maybe times 3 here-- and I'm pushed along the trajectory, it's actually going to incur just as much cost, roughly, as I'm pushed another direction. There's a quadratic bowl literally centered around x0 at time t. That's what this equation says.

And this quadratic bowl is the cost-to-go estimate. It says, if I'm away from the trajectory, I should expect the cost I incur in getting back towards that trajectory to be this quadratic form. Is that OK? And the key point is, because I've re-parameterized my equations in terms of x bar, this quadratic bowl always lives on that trajectory.

My cost function was x bar Q x bar. My best thing to do is to drive x bar to 0, which means to drive my system back to the trajectory. People OK with that imagery? It doesn't look like they are. Everybody's OK. Are we OK here the LTI stabilizer being an LQR bowl-- or a quadratic bowl? So the farther I am away in the directions defined by S, I'm going to cut some cost getting back.

This is just the same thing that says, if I'm at this point in the trajectory, I'm going to cover this cost-to-go. And the best thing to do, the minimal cost-to-go is living right on that trajectory. As a consequence, the optimal controller, which tries to go down the landscape of the cost-to-go, is going to drive you back to the trajectory.

Now, I said all that because I'm about to do something that sounds totally wacky. Would it ever make sense for me to design a slightly different cost function, which, when I linearize and design the feedback controller, I end up with a cost-to-go over here?

Let's say I have some nominal trajectory. I found, through whatever method, some reasonable system trajectory, but I really-- I'm still not happy with that. The trajectory I really wanted was something like this, let's say. Would it make any sense to do my linearization around this trajectory, and try to drive the system to this other trajectory?

AUDIENCE: You mean like scaling your optimal trajectory?

RUSS TEDRAKE: I don't even mean scaling. They could cross. They could do whatever. It's not a simple scaling. Let me give you a simpler version of the problem.

AUDIENCE: [INAUDIBLE]

RUSS TEDRAKE: Say that again.

AUDIENCE: [INAUDIBLE]

RUSS TEDRAKE: Yes. I'm going to divine a cost function, which would have it so I prefer to live on that trajectory. Let me do it in the time invariant case just so it's clear. Let's say my coordinate system's back and simple. It lives around 0. Let's say I have that cost function, or actually that dynamics.

And instead of-- my original cost function was just x transpose Qx-- let's say my cost function now is-- let's think about this problem for a second.

So let's say I have a linear system. Now, the LQR controller we did initially-- little sloppy with that. The LQR controller I did initially always assumed that the desired place you wanted to be in life was 0.

If the desired place you want to be in life is a constant-- it's 3, let's say-- then you can still do your linear quadratic regulator. Just move your coordinate system so the 3 is 0. But let's say I've got a linear system, but I want to drive it through some trajectory-- time-varying trajectory-- x desired as a function of time.

Then I can't quite just recenter the origin. I've got to think about, how do I drive my linear system through some other trajectories? Now the-- it's actually-- LTI system, but my cost function is time-varying, because my-- I have the desired trajectory that varies with time. The result-- I won't write it down again-- again, I can do this Riccati equation. Back up.

The only difference is that the quadratic bowl is no longer going to be centered on the origin. The quadratic bowl is going to move with that desired trajectory. OK? Yeah?

AUDIENCE: If that's far away from where you linearized, could you--

RUSS TEDRAKE: That's an excellent question. But this is a linear system, so first, we don't have to worry about that, but don't let me forget to go back to that. So I can drive my linear system through some trajectory that's non-zero beautifully with an LQR controller. The only problem is that my LQR controller has to have has a cost-to-go function and a controller which is not pointing me always at the origin. You wouldn't want that.

So in fact, the way it looks-- there's a lot of ways that people derive it. With Pontryagin, it's not too hard to derive. I prefer to derive it with the HJB. I'm not going to do the derivation, but-- I don't mean to bore you, but what you end up with is J of x of t has a form x transpose S of t x plus x transpose-- I call this S2-- S1 of t plus S0 of t.

It's a full quadratic form. When I just have this, it's always a quadratic bowl. It's always centered around 0. If you want it, in general, to be a quadratic bowl that's not necessarily at 0, you need the full quadratic form. I could equally well have written this as x minus x something desired, S of t. But let's work with this form. Yeah?

So this is just an equation of a quadratic bowl, not necessarily centered on the origin. And the LQR derivation gives me my backwards dynamics for S2. It gives me the backwards dynamics for S1 and for S0. And it's in the notes. It's actually already in your notes. It's in the HJB chapter that has been up there for a while.

OK, now, the reason I'm on about all this is that there's another way-- I told you about shooting methods. I told you about direct co-location. There is yet another way that people like to design trajectories, which use LQR directly. And that's this iterative LQR procedure.

OK. So let's say I have some trajectory that I've already found, x 0 of t, and I have some different trajectory, which is my desired trajectory, x desired of t. Then, using this optimal tracking-- if you stick back in the time-varying components, using this optimal tracking, I can linearize my dynamical system around that.

So I have no guarantees that x desired is a feasible trajectory. In fact, many cases-- it's not. For instance, x desired might be B at the goal at all times. If I came up with a perfectly feasible x desired trajectory, I probably wouldn't be running an open-loop solver. I want to get to I want to get as close as desired-- as possible to the x desired while potentially minimizing cost and respecting the dynamics of the system.

Here's one way to do it-- linearize my system around my initial guess, x0 of t, then design a linear optimal tracking-- linear time-varying optimal tracking which tries to regulate my system as close as possible to that trajectory. Now, what Steven said was exactly on point. If I drive my system away from where I linearized, there's no guarantee that my linear model is going to be any good here.

But the hope is that this trajectory is better-- a better guess than the one before. And you iterate, make another approximation around there, design the LQR controller, run the LQR controller that drives me here to find the new u tape. That defines my new trajectory-- repeat. OK? That's called iterative LQR. What else is it called? Do you know?

Yeah. Do you see that? It's differential dynamic programming-- almost. There's a subtle difference, which I can tell you, if you want. There's a lot of names for it. There's another guy, Bobrow-- some of you know Jim Bobrow-- he wrote this up called the sequential linear quadratic regulators.

Any four-letter acronym that ends in LQR-- if you put it in Google, you'll find something that's probably this idea. Yeah. If you put in whatever arbitrary constant in front of it, you'll probably get this idea out.

AUDIENCE: What prevents your actuator costs from accumulating from one iteration to another?

RUSS TEDRAKE: Every iteration, you're trying to minimize your actuator cost.

AUDIENCE: Right, but I mean, if you have a lot of iterations, couldn't that potentially grow?

RUSS TEDRAKE: I don't actually add to my old u tape. I actually completely replace my old u tape with a new controller which drives me to the system.

AUDIENCE: Oh, OK.

RUSS TEDRAKE: So there's no worries about additive actions. It actually tells me in my original non-linear system what's my best guess as a u tape that goes there.

AUDIENCE: Is this basically a trick to get rid of the slow [INAUDIBLE]?

RUSS TEDRAKE: So very, very good-- so why would I want to do this? Why didn't I tell you about this first, or why-- how does this compare to the other methods? There is a sense by which-- and I thought about doing the whole derivation, but I think this-- I hope that this short discussion is sufficient.

So what I'm roughly doing is I'm using LQR to come up with a quadratic approximation of where my cost-- where my minimum is. This is very much in the spirit of those SQP methods, the sequential quadratic methods. I'm using computation on this line to come up with a quadratic approximation of where I think the new minimum should be.

So as such, it's a relatively cheap way with SQP properties, convergence properties. OK. The methods I told you about on Thursday-- the backprop through time, the RTRL-- they computed J over my trajectory. They computed partial J, partial alpha over my trajectory.

They did not ever explicitly compute the second derivative. I never computed partial J, partial alpha, partial alpha. To explicitly do an SQP update, somebody needs to compute the Hessian of that optimization. I'm relying on SNOPT to do some bookkeeping to estimate the Hessian to do the second-order update.

I would do better if I had an efficient way to compute the second derivatives, and I could hand that directly to SNOPT or whatever, and we'd get-- expect faster convergence. This isn't quite the gradients that I want, but it has that feel to it, and it has similar convergence properties. So what you should think about is you should think about this is a more explicit second-order method for making a large jump in my trajectories with sequential quadratic convergence results.

I feel like I've lost everybody now, but ask questions, if you need to. The advantage of it is that it's fast. It could potentially require very few iterations to converge. One of the strongest advantages is that there's no explicit way to do constraints. You have to think harder about how to do constraints in this.

And I know less formal guarantees that it will succeed, because it's an approximation of that quadratic. So the RL community uses DDP a lot, and actually, a lot of people who do DDP do iterative LQR, for instance. For instance, Peter [INAUDIBLE] and those guys-- they always call DDP. They're actually doing iterative LQR.

DDP explicitly actually has-- you have to do a second-order expansion of your dynamics, so you don't just get A of t x. You actually go to second-order expansion of your dynamics. So it's a little bit more expensive of an update, but most people equate it almost exactly to iterative LQR.

AUDIENCE: So this x0 trajectory, this isn't a trajectory you found by doing RTRL or something like that? This is something different?

RUSS TEDRAKE: Good-- so this could be a standard replacement to RTRL. I could start with a random x0 trajectory. So maybe it's better to start with a random u trajectory, simulate it, and get an x0 trajectory. And then it will quickly reshape until it gets as close as possible to this x desired trajectory.

AUDIENCE: But you're reshaping your control actions that get you to the x trajectory?

RUSS TEDRAKE: Yes. So I'm reshaping u, resimulating to get the new x. Yeah. I wrote it more carefully in the notes, and-- but I hope this is the right level to do the class. And there's one extra thing that-- so I say this works if you have a desired x-- desired trajectory, which means your cost function has this sort of a form.

The advocates of iterative LQR and DDP say that every cost function has this form. This is just a second-order Taylor expansion of whatever non-linear cost function you want. So write down whatever non-linear cost function you have, do a second-order expansion on it, and you end up with a quadratic cost function like this.

And you can then approximate that solution with an iterative LQR scheme-- or RTRL, or backprop through time. This is the third out of our list of methods. My goal is only to know-- so that you know that it exists. And you can read the notes if you want more, and you can read the papers if you want more. OK? Yeah, Michael?

AUDIENCE: So I think last time you talked about you're parallelizing the deviation from your non-control input. So what if you were-- like as you iterate the controller, [INAUDIBLE]?

RUSS TEDRAKE: Good-- the total cost is actually the cost with respect to some u desired. So I end up trying to optimize that in a coordinate system based on u0, but the cost I'm trying to minimize is the u the original coordinate system minus u desired-- which, in a lot of cases, is 0.

Although I do it in a weird coordinate system, and it actually eventually subtracts itself out because I add it back in at the end, and-- it's quite easy to, for instance, minimize u squared in the original coordinate system. OK? So on Thursday, we get to do walking robots. We're going to move on to the next major thing.

But you've now learned three of the open-loop trajectory optimizers that people really use-- iterative LQR very quickly, RTRL backprop through time-- I grouped as one-- the shooting methods and direct co-location. There's another one that's recent addition to the scene, which is this discrete mechanics and optimal control, this DMOC.

If anybody was excited about that and wanted to do a class project on that, that would be a perfect thing. Grab that paper. Show us that it works on the [INAUDIBLE] carpole. That'd be beautiful. I'd love to have that-- have us try that and see how it compares to the other methods. You've got a pretty good toolkit for optimal control now-- practical optimal control.

And it works for flying robots, but it also worked for your wheeled robots, if you want to control them with better control. You could do a drop-in replacement LTI optimal tracking controller, and it would be better-- assuming your model's better. So you have these tools.

Quick procedural things-- I know we're out of time. So next Thursday-- well, so let me say the good thing first. In two weeks, you're on spring break. Yeah. The Thursday preceding that is our midterm. We haven't had a midterm in the class before, so there's no old exams for me to give you, but John and I are going to try to come up with some representative problems for you to take home for Thursday of this week so you can have some problems to munch on over the weekend. It'll be an in-class exam Thursday before spring break, which is a week from Thursday. OK?

AUDIENCE: [INAUDIBLE]

RUSS TEDRAKE: Yes. So open-book-- well, you can grab whatever notes-- open-note exam-- absolutely. Well, I'll say it more in the preparation package, but roughly, we're going to-- I think, if you have your notes with you, if you've done the problem set-- and most importantly, if you know how these algorithms-- where the algorithms relate to each other and where they'd be used in different systems-- I can guarantee I'm going to ask you something about that-- then it's not designed to be a killer.

Good-- and I hope you start thinking about projects. Just out of being a fairly nice person, I wasn't going to ask you to do projects before your midterm. But this time last year, I was asking people to submit project proposals. We're going to do that immediately after the midterm.

If you've been chewing on, this method looked like a really good match to my research problem, or I've never actually thought about juggling robots before, or something like this, you can imagine-- so in the fairly near future, we're going to ask you for a half-page project proposal that we can iterate with you on to get going on a world-class final project. Yeah? See you Thursday.