MITOCW | Lecture 15 | MIT 6.832 Underactuated Robotics, Spring 2009

PROFESSOR: OK, so we added another tool, quickly I admit, last time to our arsenal, saying that if you've got a system where you're trying to do a trajectory plan for and your trajectory optimizers are failing you, because they're only guaranteed to be local, locally good, then there are class of more globally more complete algorithms that are guaranteed to find a solution, if it exists, based on feasible motion planning. So we talked about ROTs mostly. I talked a little bit about also the more discreet planning, A*STAR and things like that.

OK, so, so far, our methods are still clumped in very distinct bins. We still have our value iteration type methods, our dynamic programming methods, which I love, which give us policies over the entire state space, so global policies. But they're stuck by-- they're cursed by the curse of dimensionality. So it only works for low dimensional systems.

OK, we've been talking also about-- and we've been talking about policy search in general. And I'm going to, later in the class, make a point that that's not just about designing trajectories. I made it initially. We'll make it more compelling later.

But mostly what we've been talking about other than that has been falling under the class of trajectory planning and/or optimization. OK. And this is only locally good but scales very nicely to higher dimensional systems.

So you might ask, how well does this scale? I don't really think there's a good limit. I mean, it just depends on the complexity of your problem. People have used ROTs very effectively five years ago on 32-dimensional robots. That's pretty darn good, right? If I have a system where the start and the goal can be easily found, Alex says we can do it in thousands of dimensions.

If I have a system where the only hope of-- if I have a six-dimensional system where the only hope of finding my way from the start to the goal is by going through this little channel, then I told you that's going to fail, even in low dimensions. So it's a hard question for me to specifically say what class of system should you expect this to work for. But I think they're the best tools we have for higher dimensional systems, OK.

So the big question I want to address today is whether these ideas, which seem very local-- we were talking about single trajectory planning-- can be used to design a feedback policy that's more broadly general, that's valid over lots of areas of the state space, OK. Does that makes sense, what I'm saying? Yeah? I'm saying I could design a single trajectory, but that's really only relevant very close to the trajectory.

So let's make our favorite picture. Let's say I've designed for the simple pendulum a nice trajectory, which goes up and gets me to the goal. And that's good. If I start here, I know exactly what to do. We talked about stabilizing it with LTV LQR. So that means if I start here or here, I'm in pretty good shape.

If I'm smart enough to index into the closest point on the trajectory, then maybe even starting here, it's fine. I'll just execute the second half of that trajectory. But what happens if I start over here or if I start over here? Probably, the controller based on the linearization is not going to have a lot to say about the points that are far from my trajectory.

So the goal today is to take these methods that we've been pretty happy with for designing trajectories, and even stabilizing trajectories, and see if we can make them useful throughout the state space, just to see how well that can work. OK, there's a couple of ideas that I want to get to, but first, I want to make sure I say that there's no hope of getting-- there's no magic bullet here. So there's no hope of me finding global optimal policies, unless I'm willing to look at every state/action pair.

I'm not going to tell you that I can use these trajectories to just magically do what value iteration did in high dimensions. That's not what I'm saying. Unless you have some analytical insight which turns the problem into a linear problem or something like that, I'm not saying that I'm going to give you globally optimal policies. What I'm trying to say is we can get good enough policies, potentially, using these methods, OK.

So I just want to make sure I make the point that we really can't expect globally optimal policies unless we explore every state/action pair, of maybe if we have some analytical insight. OK, so the curse of dimensionality is real. It's not that some-- the value iteration algorithm is a little quirky. It's got this problem of dimensionality. It's really not that at all. It's not that somebody hasn't just come up with the right algorithm.

The problem is you can't know if there's a better way unless you look at every possible way. That's the real problem. I mean, so it might be that I want to find my way from the start of the-- front of the room to the back of the room, and I've got some cost function which penalizes for the number of steps I take.

But unless I go down that third row, I didn't know that there was actually a-- see if I can say something not ridiculous, but some pot of gold or something in the middle of the third row. And I just didn't see it, and I'm never going to see it unless I go down the third row. So you really can't get around that.

So the goal is to really-- maybe we can efficiently get good enough policies. And I don't care about optimality, per se. I've said that before. I just care about using optimal control and the like to turn these things into computational problems.

OK? So there's a couple ideas out there that are relevant. The first one sounds a little silly, but it's increasingly plausible. Let's say my trajectory optimizers or my planning algorithms got so fast, or maybe just computers got so fast that I didn't have to do any work in the algorithms, that it takes me a hundredth of a second to design a trajectory from the start to the goal here. I've got a real time execution task here. Every, let's say, hundredth of a second, my control system's asking me for a decision about what to do.

But if I can plan fast enough, and I find myself in this state, then you could just plan again. You could really just, every time you find yourself in a new state, plan a trajectory that's going to get me to the goal. If I find myself-- so if I'm executing this trajectory and I get pushed off on a disturbance, no problem. Every step, I'm just planning a trajectory to the goal.

If you can plan-- if we teach the course again in five years, maybe that's the only answer. I don't know. If you can plan fast enough, that really is a beautiful answer. For the most part, the problems we've looked at so far are not that easy that you can plan that fast, but there's a middle ground.

So this was basically plan every dt. There's a middle ground that people use today a lot. I mentioned it once before. But a lot of times what we do to make real time-- to make the planning fast enough to execute in real time is a lot of times we'll do some sort of receding horizon problem. So how's that going to work?

The simplest answer is, for receding horizon, I've got some long-term cost function, and I've got my total cost function is from t equals 0 to some t final g of xu dt. I could-- I did it discrete time. That's fine. So n to capital N for discrete time.

And let's say it takes me too long to plan N steps ahead, but I know I can plan three steps ahead really fast. So a lot of times people will actually approximate that with the problem of just looking some finite receding horizon step ahead. And if you can-- if you're doing it at every ti-- if at time 2, you're asking for the receding horizon plan, then you can just look from time 2. So let's say my current time to my current time plus 3 gx of u.

That could be an arbitrarily bad estimate of my long-term cost, of course. If you're clever enough to have a guess at the long-term cost, then you can put in some sort of estimate of what j x from t plus 3 might be, and that's going to help.

So for instance, let's say I find myself off my trajectory somewhere over here, and I'm willing to say my planner's fast enough. My controller's running at 100 hertz. And in a hundredth of a second, I can about solve an optimal control problem that's of half a second in duration, let's say. That's a reasonable thing.

Half a second along puts me-- it would put me here. So let's say I'm going to design-- I'm going to use my planner to design a trajectory that gets me back to this in half a second. And then I use my cost to go that I already knew from this design to get me to the goal. That's one way to implement what I just said, OK.

And it's not just talk. I can show you a good example of it. So I showed you guys this once before, but let's just look at it again quickly. This is Pieter Abbeel's and Andrew Ng's work on the autonomous helicopters, OK. So they execute these comically cool trajectories with their helicopter. The way they do it is actually, they get a desired trajectory from a human pilot, and then they stabilize that in real time.

They do-- he calls it DDP, but it's actually what we've been calling iterative LQR. I told you that a lot of people blur the lines, unfortunately, between those two. So they do an iterative LQR controller design, and they decided that it's fast enough that they can do it three seconds into the future.

So they're doing exactly these receding-- every dt for that control for that helicopter, they're doing iterative LQR to design a trajectory that's going to get me back to my pilot's trajectory. And they're running it every dt, thinking three seconds ahead, and they say, that's comparable to the time of-- the dynamics of instability for their helicopter. Yeah. Put it all together, and you get this thing tracking pretty cool trajectories, OK. It took a lot of good engineering behind that, too, of getting the model right and getting the helicopter right, but it's pretty impressive.

OK, so if you can plan fast enough-- and like I said, in a few years, maybe the planning algorithms are going to be-- and the computers are going to be so fast and the planning algorithms are going to be so fast that we never do value iteration anymore, but I kind of doubt it. I think that there's always going to be reasons to do more global methods. If you can plan fast enough, even a little bit into the future, that it might be good enough to just turn your planner immediately into a feedback policy.

OK. We don't do that so much in my group. I think it's a good idea, and it makes sense. But I do think there's a lot of other good ideas out there on how to turn your planners into policies. OK, the next one is multi-query planning. Anybody know what I mean by that?

AUDIENCE: [INAUDIBLE]

PROFESSOR: No, that's not what I mean. You can imagine doing something like that and it meaning this, but-- so I spent relatively little time on the ROTs, but actually, it's one of the tools we think a lot about in my group now. It's actually-- the only reason I spend little time on it is I think that seeing the big idea in class is enough, that the ideas are so simple, that when you do your problem set and make it work, that's the best way for you to learn about it, OK. So it's such a simple idea, and it just works very well.

OK, so let's say we've got these ROTs that we like, we know and love. And for the pendulum, I showed you a plot of the ROT trying to find its way to the goal. It started splintering off lots of-- eventually, it'll find some trajectory that will find its way there, but along the way, it's generated lots of trees that do random things and lots of paths that didn't turn out to be useful.

And what you have is a web. In this case, it's a tree. If you run the ROT once, you have a tree of feasible trajectories that you could execute on the real robot. It happens that one of them got me from the start to the goal in my initial problem formulation.

OK, but instead of throwing all that computation out and just keeping the nominal trajectory, I might as well store it. If I get a new problem, which is, let's say I wanted to start from here, like I said and get to the goal, then really, all I need to do in my new time, in my new planning problem is connect back to my old solution. If I can find a new plan that gets me back here, then I can just ride the rest of the solution into the goal.

Simultaneously, if someone were to tell me I want to get to a different goal-- let's say I want to get the system to the upright with some velocity-- all I really need to do is find a way to connect from my old plan to the new goal. So as you design these, the first start to the goal planning problem, where you're designing trees that try to get as-- cover all over the place, could be potentially very painful. We might take a long time to find your way from start to goal.

But if I want to solve a new problem which is not so different, then it could be actually very efficient to reuse your old computation and do a multi-query-- this is a multi-query planning idea, OK. And I think that idea is so good that it's actually-- if you do this again and again, you're going to slowly end up with this web of feasible trajectories that you could execute. People call it a roadmap. When you have some network, some graph of these feasible trajectories, people call it a roadmap.

If all you care about is getting to the goal, then all you need to do is connect to your existing roadmap and write it to the goal. If your roadmap is so rich that once I connect to the roadmap, there's actually a bunch of different options, bunch of different paths I could take through the graph to get there, well, then at least you've got a discrete planning problem, and you can do A*STAR on it or something like this. And effectively, the trajectories I've already generated will turn this back into a discrete planning problem.

That idea is so good that some people believe it's the only thing you need to do, OK. There's a camp out there that does these probabilistic roadmaps that's-- Jean-Claude Latombe, I think, is the head of the camp, started these ideas. And they believe that you should address a complicated motion planning problem in two steps. First you'll construct some dense enough graph, a roadmap, I guess I should call it. And then once you've got it, you just do your query phase, OK.

So let's think about that in a configuration space. I've got a bunch of obstacles, and I want to get myself from some start to some goal. All right, if I know I'm going to be doing a lot of these things, then it actually makes a lot of sense for me to go ahead and build a pretty good graph. So before I even start to solve the first problem, let's just drop in a lot of random samples throughout the space, choose uniformly, OK, at the space. Every time I add a point in the configuration space world, they try to connect that new point to the end closest points with simple strategies.

So I'll pick a point at random, I'll try to find the guys that are close to it, and I'll connect with it. Pick a new point at random. Oh, there's really only one guy close to it. I'll connect to it. Pick another point. Maybe these guys are connected. And that's it.

And if I do it enough, and then I come up with a pretty good roadmap-- maybe this guy was the one that connects to everybody-- that when the query phase comes along, again, all you need to do is connect to your roadmap. I got a new query. I just connect to my roadmap. I do whatever my discrete searching problem may-- A*STAR or whatever to find a path from the start to the goal, OK.

I actually think it's a very beautiful idea to have this web of possible trajectories covering the state space. And then all it takes at execution time is connecting and then executing your trajectory. Now the probabilistic roadmaps, again, this step of connecting nearby points in under-actuated systems might be hard. Might be as hard as finding the path from the start to the goal.

So maybe what you do here is actually do a [? DR call ?] or something to find that path, or you do an RRT, or one of any of the other methods we've done to make these initial connections. And maybe to make them feasible to execute, you've got to do some trajectory stabilization to get on that. But if you can solve some local planning problems, then you can use these big roadmap ideas to maybe do more global behaviors, OK.

So again, I think multi-query planning is a nice way to go from local policies to more globally valid policies. Yeah.

AUDIENCE: I can see that working pretty well with a static obstacle field.

PROFESSOR: Good.

AUDIENCE: Could it move [? moving ?] obstacles, and might-- the roadmap might change?

PROFESSOR: Well, I don't really know what the proponents would say. But if you know where the obstacles are, then-- or if you even sense where the obstacles are going to be in a receding horizon quickly, then you could-- maybe this one's blocked, and I can just take another path. But if I have a rich enough road map, hopefully you can get around that.

And the other thing is, if I have a model of how those obstacles are changing, then naively, that just adds one dimension in time, let's say, to my plan, and I just have to do a higher dimensional plan. But I think the case you're thinking about is if these things are just moving on their own. I don't have any good model. I suddenly find that I'm obstructed. Then, again, you could dynamically replan or you could-- by either taking a different path here or making a new edge if you had to. I don't think it breaks the fundamental goal.

You could almost think of this as having-- in a dynamic sense, you could almost think of this as having a bunch of repertoires, a bunch of things I know how to do. So maybe if it's a walking robot, maybe I know how to take a step here. That's one of my edges. I know how to execute that. I know how to take a big step. I know how to take a small step.

It's a repertoire of local skills, local trajectories in this case. Then I just got to stitch them together in the right way. So that's a fairly robust thing, even if-- yeah.

AUDIENCE: Given a rich enough roadmap, would you have problems finding-- choosing the best path among those path nodes, like discrete search?

PROFESSOR: The discrete search, I think in general, you should think of as being unlimitedly-- basically unlimited. I mean, compared to all these continuous time methods, it's very, very efficient. People doing it on huge collections of nodes very efficiently, especially if you can do A*STAR, if you find a good heuristic. I mean, this is how you can go to-- to maybe overplay the title, this is how you go to MapQuest and you ask it to go from Boston to California, and it just happens. These things are very fast, even with a lot of nodes, a lot of roads. Yeah. Yeah.

AUDIENCE: How did it compare to [INAUDIBLE] discretize in state space?

PROFESSOR: Good.

AUDIENCE: [INAUDIBLE]

PROFESSOR: So this-- very good question. Let me answer that first, and then I'll-- yeah. So I almost talked about this last time right after I said, what happens when you turn this state space into buckets, how it's a reasonable thing to try but not very elegant. I think these guys would have put this topic immediately after that, saying, instead of discretizing in some unnatural grid maneuver, we're discretizing here by sampling randomly.

That has the benefit that you could, for instance-- you can actually-- you don't have to sample uniformly. Maybe you care more about things in this area. You can bias your sampling distribution, same way you can add more grid cells or something like that. But the real benefit is that it's a more continuous process. It's not stuck in some very discrete bins. OK. Sorry, second part of your question.

AUDIENCE: Well, now it would follow, like if we have a discrete [? board ?] and we can run any of those algorithms on [? value ?] [? iteration ?] on top of it, [? we can ?] [? find out? ?]

PROFESSOR: Yes, exactly. So why did I say A*STAR. I should have said value reason on the-- right? Yeah. Right. I mean, A*STAR can be faster than value iteration. If you have a good heuristic, you don't have to-- yeah.

AUDIENCE: So when you're doing the sampling here--

PROFESSOR: Good. Yeah.

AUDIENCE: --you were-- ROTs do this very uniform sampling. And you say you can bias the sample.

PROFESSOR: Yes.

AUDIENCE: But if you're doing this before you even do your first path, why don't you actually choose [? optimal ?] [INAUDIBLE]? Can't you do some kind of [INAUDIBLE] diagram with this and just say, I'm going to test using the best I can find, given that I have a model of the world, and then [? get them ?] [? with sampling, ?] [? your ?] [? subcontinuous ?] time, and then you find-- why is the sampling [? still ?] part of this--

PROFESSOR: Excellent point. I think if the problem permits that, then you should absolutely do that.

AUDIENCE: OK. So then--

PROFESSOR: I think even for the pendulum, though, I wouldn't know how to tell you what the optimal sampling is, because the way these things connect are non-trivial. They're subject to the dynamic constraints.

AUDIENCE: Right.

PROFESSOR: Right. So if you could formulate that and solve it for this-- and maybe you can. Maybe people have. I don't know that. I haven't seen that. But then that sounds like a very reasonable thing to do.

AUDIENCE: But it doesn't have to be quick, right?

PROFESSOR: It doesn't have to be quick--

AUDIENCE: [INAUDIBLE] first time--

PROFESSOR: No.

AUDIENCE: --can be as slow as possible--

PROFESSOR: It could--

AUDIENCE: --because you want it-- well--

PROFESSOR: Well, right.

AUDIENCE: --times the universe can explode with that, but--

PROFESSOR: Right.

AUDIENCE: OK.

PROFESSOR: Right. Take a chance to-- this is, explore your system. Build things that are good in random places, and then worry about connecting them later. Mm-hmm. Really good. OK.

So again, making these connections in under-actuated systems is more subtle. It might be that there's a lot of one-way connections, but we can still do-- we know how to do graph search, OK. But these are generally good tools, and they've been used a lot in robotics lately.

These are-- the other ones, the Rapidly Exploring Randomized Trees, goes by RRTs. These go by PRMs, Probabilistic Roadmaps. A lot of people seem to think that they're competitors, intellectual competitors with RRTs, and I don't think that they are really.

I think the RRT guys would just say, well, you just use an RRT to make the connections, and the roadmap is still a very good idea. And I think RRTs effectively make roadmaps. So I think they're very harmonious ideas. Excellent.

So that's at least two ideas to take these local trajectory optimizers and turn them into more of a feedback policy. But there's a big one, big one that I like a lot, that I haven't said, OK. So big I'm going to go back to the left.

OK, let's say it's idea number three-- these things aren't perfectly orthogonal, but this was the breakdown I was most happy with, OK. Feedback motion planning. OK. So, so far, we've talked about building some trajectory that we thought was good, and then afterwards, go through and stabilize it with feedback. That's not always the best recipe, because you could imagine, for instance, designing a controller that locally looked very good but was completely unstablizable.

I go to then-- I'm done with this. I say, perfect, my first stage of my control design picked this trajectory. Now I'm going to run LTV LQR on it to stabilize it.

And then I find out, whoops, right there it's not controllable or something and that my cost to go function blows up. Maybe my open loop trajectory optimizer told me to walk along the side of a cliff and wasn't really paying attention to the fact that stabilizing that's hard. Or maybe it was saturating my actuators the entire time-- that's a very real possibility-- and left me no margin of control to go back and stabilize it. OK.

AUDIENCE: [INAUDIBLE] [? putting ?] those edges down, that it is actually feasible to go from A to B, so--

PROFESSOR: Yeah. It's definitely feasible to go from A to B, but it doesn't say that-- nothing thought about whether if I get disturbed epsilon from this, whether I can recover.

AUDIENCE: So you're worried about the noise.

PROFESSOR: I'm worried about noise, right. So it's feasible for me to walk along the side of a cliff, but I wouldn't want to be bumped. If I know I'm going to be bumped, then I pick a different path, OK.

So you can imagine-- for maybe each of those examples, you could imagine ways to try to make the planning process more-- let's say, OK, well, don't use your full torque limits. Use 90% of your torque limits. That's a good idea. That'll help.

But there's a more general philosophy out there, which is that you shouldn't just do trajectory planning and then stabilize it. You should really be planning with feedback, if that makes any sense, OK. Well, it'll make sense in a minute.

There's a lot of ways to present this. I thought the best way would be to start with a case study, someone who-- a problem where people really use this, OK. There's been a lot of people that have been interested in making robots juggle. One of them's been sitting in the room here. The ones that did a lot of the work I'm talking about here is Dan Koditschek's camp. OK.

So it's actually very, very harmonious with John's lecture on running, and that's why Koditschek's done both, for instance. Now let's think about the problem with making a robot juggle, OK. So the first thing you need to think about-- and let's make a one-dimensional juggler, OK. So we've got a paddle here, constrained to live in this plane, and we've got a ball, also constrained to live in that plane. Yeah?

And your goal is to-- if this thing is in a rail, it can only move vertically, your goal is just to move that paddle to, say, stabilize a bouncing height. Let's say you've got a desired height. OK. This is the 1-D juggler. I think they call it the line juggler by Martin Buehler and Dan Koditschek. Martin went on to build Big Dog at VDI, and now he's at iRobot. So these are famous guys, OK.

So the dynamics are pretty simple to write down. You have a mass of the ball. You have some dynamics of your paddle. You assume that the mass of the paddle is much, much bigger than the ball. That simplifies some things.

And so now the dynamics are just ballistic flight of the ball. You need some trajectory. Your control is to design some trajectory of the paddle, and then you have an impact dynamics, which these guys use an elastic model-- model it is an instantaneous elastic collision with a coefficient of restitution. That's a reasonable collision model if your energy is conserved.

And again, they assume that when the collision happens, the ball changes direction and keeps 90% of its energy, and the paddle was unaffected. Relative to the mass of the paddle, the ball is negligible.

AUDIENCE: [INAUDIBLE] juggling the balls are almost completely [INAUDIBLE].

PROFESSOR: That's true. These are, I guess, not-- [? Philipp's ?] are completely almost as hard as possible. In his project, he said he spent lots of time trying to find the perfect ball, which was the perfectly machined, very hard precision ball, yeah. Compliant juggling, maybe that's our next challenge for robotics, squishy balls.

OK, good. So it turns out they do a really nice control design. It turns out to be very natural to-- the controller that they come up with for the paddle uses a mirror law. Turns out if you can sense the state of the ball and you just do a distorted mirror image of that ball, then everything gets really easy.

Your impacts always happen at 0. It's at the same place. And you can, just by changing the velocity here, you can roughly affect the impact height.

So what they do is they can nominally stabilize some limit cycle with just mirroring the ball, and they add an extra term to stabilize the energy to get it to whatever height they want. So they do a distorted mirror image of ball trajectory plus-- the distortion is scaled by some energy correcting term. It's a beautiful thing. Very, very simple controller.

Has a nice, very stable solution. In fact, I think they prove it's globally stable for-- you can tell me if-- is it globally stable in the 1-D case? I think it probably is.

OK. How do they prove it's globally stable? They do an apex to apex return map. And the same way we did for the hopping models and all the other models, these guys were pushing the unimodal maps and getting some global stability results out of that. That's why I think that they had a global result, OK. So it's actually exactly like a hopping robot. Just the ball's moving instead of the robot.

OK, so they got a pretty good controller for 1-D juggling, and then they started doing 2-D juggling. I think I have the vi-- I don't have the video for the 1-D juggling somehow, but I do have the video for the 2-D juggling.

Yeah, so here's your 2-D case showing off doing two balls at once, since all that matters is the state of the robot when the impact occurs. So you might as well do something else during the other time, like stabilize another ball. And you can see that actually, it turns out to be pretty easy to get stability in this plane, just because if you tend to-- if you're too far to this side, you tend to get hit earlier, which causes you to go out more and vise versa. So that stability almost comes for free.

AUDIENCE: [INAUDIBLE]

PROFESSOR: This is, I think, vision off to the side. I know it's vision off to the side, where they're tracking the bright yellow balls. If it had been dark gray balls, it might have been something else. But the bright yellow tennis balls suggests vision. Yeah.

And they went on to do the 3-D juggling. This one was, I remember, in the basement of the Michigan AI lab when I was there, behind that curtain. It's always using the vision sensing for the balls. You could do you pretty good things.

And then they got so good that they started doing other maneuvers like catching and palming and things like this, OK. It actually turned out to be the same, pretty much, control derivation. They just set the desired energy to 0, and suddenly they have a catching controller.

And then this is palming when they're doing their thing, OK. And then they can get it back up to catching with the same sort of energy shaping. And I should show you, you don't actually need all that feedback to do it. You don't need to sense the ball.

Here he is. This is [? Phillips. ?] We'll show the one where he's pushing it so you can tell who it is here. Blind juggler. So this is open loop stable juggling.

You can see-- actually, do you see the ball up there? Yeah, it's going to a stable height, and he's moving it around. He's got just a itty little bit of concavity in that plate, which gives it all the passive stability properties. And you've got versions where it's doing things off to the side in 3-D or in 2-D and-- yeah, so let's open loop stable.

So juggling is actually a really cool problem for robotics. It's led to a lot of nice dynamic insights and party tricks, I guess. Yeah.

OK, so these guys said, we got pretty good at juggling. We can do a mirror law to stabilize whatever juggling height we want. We've got a catching controller also, which has roughly set the energy to 0 and just sort of does this step. And they also had a palming controller, which was when the dynamics were actually on the paddle, they did a little bit of different things to be able to move it around without it falling off the paddle.

What they were left with was this challenge. And we've got these controllers, which are good locally. What do we do to make them do more interesting things? So they actually-- so if they want to transition, for instance, between the catching and the palming-- the bouncing and the palming, they use their catching controller. Maybe they want to avoid moving obstacles. They want to do multiple balls. They want to do all these things.

They introduced a really nice, beautiful picture of feedback motion planning using funnels. OK, so every one of those controllers had the property that it would take initial conditions in state space and move them to some more desirable state. So for instance, the ball hopping, the ball could be anywhere here. By applying this controller for some finite amount of time, when it's done, it's going to be closer to its apex height.

In many cases, not in the juggling case, not in the experimental juggling case-- and even in the model one, I guess they do. But in many cases, you actually have Lyapunov functions which describe the way that convergence happens, OK, but it's not strictly necessary.

So the idea is, let's think about this thing as a funnel, OK. It takes lots of states in. So this is initial states. And after applying it for some finite amount of time, I'm going to be some-- you get some new final states. And if my controller was any good, then hopefully the final states are a smaller region than the initial states.

So in some sense, this is a geometric cartoon for a Lyapunov function. Lyapunov functions take my state in, and descend down, and will put me in some other state, OK. Experimentally, you can also find these things, even if you can't do the Lyapunov function. Experimentally, this input is basically the basin of attraction of my controller.

So if it was really a basin of attraction and it stabilized some fixed point, then if I ran it long enough that it was asymptotically stable, I'd call it a basin of attraction. Here I'm just going to run it for some finite time. So you have to be a little careful calling it a basin, but I think it's still intuitive that this is the-- there's lots of names for this.

Another name for it is pre-image in the motion planning world. Lot people call this the pre-image of our action. This is, I guess, the post-image, yeah. These are the set of states where my controller is applicable.

I'm going to have a funnel for the mirror law. I'm going have a funnel for my catching controller. That takes a different set of initial conditions and gets me where I want to be. And I have a funnel that can allow my palming to do different things, OK.

And I might even have lots of funnels. So I might have a different funnel given the mirror law where my desired energy is 4, versus the mirror law where my desired energy is 6. Maybe those should look like different funnels.

So the picture that these guys gave us-- this is Burridge, Rizzi, and Koditschek, the guys. Al Rizzi's at Boston Dynamics also-- is that you can do feedback motion planning as a sequential composition of funnels, yeah? So if I want to get from one state to another state, and I don't have a single controller that will get me there, all I need to do is reason about a set of these-- a sequence of these funnels for which the first funnel takes me from my initial conditions into a domain where my second funnel is applicable. And then I can use my second funnel to get me somewhere else, and then my third funnel maybe will get me to my target.

So if my goal state is somewhere abstractly here in state space, that's not accessible from any one-- I can't get from my initial condition to my goal with any one of my controllers. I can sequence these controllers, just making sure that the output of one funnel is covered, completely covered by the input of the next funnel. Then that's enough, then, to turn this again, to use these funnels as an abstraction to take away the continuous problem and give me a discrete planning problem, which just says I just need to go through this funnel, through this funnel, through this funnel, and I can get to the goal, OK.

So in this case, they did tasks like there was a beam here that was-- they were bouncing on one side. They wanted to be bouncing on the other side. So I think they had one controller that went over the top. Then the beam got taller. They had another one where it caught it, brought it under, started paddling again. And these things just fall naturally out.

This could be the catching. Or this could be the-- yeah, catching. This could be the palm, and this could be my mirror again. And I'm right back to where I want to be, OK. Very, very beautiful idea.

As far as I could tell, everybody who read that paper was enamored by it, and nobody's really used it that much, because there was one critical problem. Figuring out what those funnels looked like are really hard. So really, the only issue, I think, is that describing the basins of attraction, let's say-- so if you read the Burridge, Rizzi and Koditschek paper, you'll see a ridiculous number of scatter plots where they put the ball in this location, they ran their controller for a while, and they determined experimentally whether it was in the basin of attraction of this controller. Yeah?

Ouch, right? That's not what I want to do with my time. So if you're willing to do that, then it's a workable method. But I think today we've got a better way to do it.

And my group, we've been working on an implementation of this feedback motion planning idea, which is very much in line with the things we've been talking about so far, which we've been calling the LQR trees, OK. And the big idea that happened is that these guys in LIDS, Pablo Parrilo-- anybody know Pablo? And Alex [? McGretsky's ?] the one who taught me about this-- have figured out new effective ways to computationally estimate basins of attraction of some classes of controls, OK.

OK, so this is a new thing. People have been doing algorithms to design Lyapunov functions for at least a decade. But I think they got really practical a couple of years ago in Pablo's thesis, actually. Think it's two Ls, yeah.

What Pablo did in his thesis is he promoted this sums of squares programming. In fact, you can even download SoS tools from his-- as a MATLAB package from his website to do this. Sums of squares programs are efficient ways to check whether a polynomial function is negative definite, OK, potentially with free parameters, and so on. These can be made-- and these can be vector variables, can be made uniformly negative definite or semidefinite, OK, or trivially positive semidefinite.

Seems like a little-- you can see how it might be relevant, OK. So this is just a mathematical idea to turn the problem of checking the positive definiteness of a polynomial into a linear matrix inequality and then a convex optimization problem. So I'm not going to go into all the details, but know that there's these tools out there that use convex optimization to check that property of a problem, OK.

And you can read more. I've got links if you want to read more about that. What that allows us to do now, at least in the case of the LQR design we've worked it out, it's possible to now check whether a function, a polynomial function is a Lyapunov function for the system.

Lyapunov functions have to have their derivatives going down over time, yeah. In order for a good function to be a Lyapunov function, its value had better be going down at all times. If your candidate Lyapunov function is even a vector polynomial function, then you can use this to check whether it's a valid Lyapunov function for your system, OK. So we can now--

AUDIENCE: [SNEEZE]

PROFESSOR: Bless you. So I threw this one in without saying it before. The only caveat is you have to take your nonlinear system and make a polynomial approximation of it, a Taylor expansion of it. It doesn't have to be first order. That's the linear system. But it has to be polynomial, OK.

So suddenly, it turns out that for the LQR systems-- remember, our value function. Let's just think of the LTI LQR. The value function turns out to be this quadratic form. It's the optimal cost to go.

That's a Lyapunov function. J of x for the linear system is a Lyapunov function. As I take control actions, my cost to go is only going to go down. It had better, otherwise it's not the optimal cost to go.

So OK. If I have a nonlinear system, where I've linearized it and done LQR control, then I expect that to be-- this function to be a Lyapunov function over some domain where the linearization was good, and eventually, to no longer have this nice negative definiteness property. Does that make sense?

The optimal cost to go from LQR isn't a Lyapunov function for the entire state. It's always going to descend for the entire-- for any initial conditions for the linear system. But when I've linearized the system, I expect this to be, J star to be, a valid Lyapunov function near the linearization. You guys should stop and ask questions now if you have any questions.

Does that make sense? I never actually said before that you can think of these cost [? to goes ?] as Lyapunov functions, but that's a nice connection between the optimal control and the stability theory, OK. But the cost to go actually is a Lyapunov function.

Remember, when we're taking a linearization doing LQR, we already know that the basin of attraction is going to be something finite. We talked about that at the Acrobot. I do a linearization around the top. I know if I'm near that point, it's got some small finite basin of attraction. If I'm inside that region, it'll go to the goal. If I'm outside that, then the linear design, controller design, isn't valid for the nonlinear system. Eventually, you're going to get far enough away that it's not going to work.

It's going to-- I think I even showed a simulation of it doing something crazy. So what we did was we designed a controller that got up there, and then we turned on the linear controller. All was good.

So a different way to say that exact same thing is that at some point, when I get too far from the fixed point, if I evaluate this function and look at the time derivative of this function, it's no longer going to-- my cost is not going to go down as time goes up. And in the case for the Acrobot, if I'm here and I start going this way, then I'm getting further from my goal. My cost is going up. At some point, for the nonlinear system, this function is not going to be a Lyapunov function for that system, OK.

So what we've got, thanks to Pablo and [? Sasha ?] [? McGretzky, ?] is a way to figure out exactly a-- well, not exactly-- to estimate the place where that transition happens using these sums of squares programs. My goal here is to tell you about the existence of these things, and I'm happy to push you more in that direction if you're interested, for your project or for whatever. But we're going to use this to do the feedback motion planning, OK.

So it turns out J is a scalar. My cost to go is a scalar. It turns out I can very succinctly describe the place where this-- a boundary of this function-- let me just write it, and then I'll say it carefully. I can describe a region of my system just by looking at the height of my cost to go.

This is a quadratic function. It's going to look like ellipsoids going out. If you were to draw this landscape, it's going to look like an ellipse, a parabola in high dimensions, yeah.

At some point, as I move farther from my fixed point, the cost is going to get higher and higher, OK. And at some point, it crosses some scalar value rho. So the way I want to design, I want to call my basin of attraction for this system the place where my cost to go reaches rho. And we've got a program, thanks to [? Sasha and ?] Pablo, which will try to estimate this scalar value, rho, as a scalar representative of the basin of attraction of my system.

AUDIENCE: Why would you use a particular cost to go rather than looking at what the variation is in the linearization?

PROFESSOR: That is-- so we're going to determine this by looking at the variation based on the linearization. So I could do this in a lot of different ways. I could look at boxes around my fixed point and try to design some geometry. The real basin of attraction is going to be some complicated thing, which depends on my LQR controller design and the way the non-linearity affects.

AUDIENCE: Right, but wouldn't-- so I guess what would seem more intuitive to me would be to say, look at the next highest order term in the expansion and then see how that's varying and use that to--

PROFESSOR: That's exactly how we're going to verify it, OK. So there's two questions. There's a question of what shapes are we going to try to verify, OK. The choice here is to verify contours of the cost to go function. I'm going to try to find the biggest contour of the cost to go function for the linear system.

You're asking if I could choose a different shape based on the contours. What we've elected to do-- and I think it's a tighter version, maybe, than what you're saying, but I could be wrong. There could be better ways-- is to find the biggest contour such that the next higher order terms of the linearization don't break the negative definiteness.

AUDIENCE: OK, so this is just for purposes of choosing a shape.

PROFESSOR: This is choosing my shape, OK. So I'm going to make this all concrete right now by trying to show you an example here. OK, here's a simple pendulum, which we know and love, OK. This is the phase portrait of the simple pendulum.

OK, and the green is at 0, 0, which, in this case, is my downward fixed point. The top is my unstable fixed point. My goal is to use these local trajectory ideas in order to cover-- to make all states go to them, OK.

Now all I told you so far is I know how to take an LQR problem and try to estimate the basin of attraction, OK. So that's step one, is take a linearization around my goal state, estimate-- design an LQR controller, and estimate it's basin of attraction, OK. And that looks like this, OK.

We've seen cost to go functions for the ellipsoids, or ellipsoids around the fixed point. I do a sums of square optimization to verify that this function is negative definite, which involves a higher order polynomial expansion of the dynamics in this form. And I try to find the biggest contour for which that system is still negative definite. That's all the detail. All you really need to know is that I can estimate now, with convex optimization, the basin of attraction of that system.

This is Koditschek's funnel at the top. That blue region is the beginning of the funnel. In this case, I'm going to run it infinitely long, so it's going to eventually get to the red point. That's the output.

OK, now how do we design funnels that try to fill this space? OK, my proposition is that we should do roughly what the RRTs are doing and start growing out to try to cover the space in lots of different directions, OK. The only difference is, every time we grow out in random directions, I'm going to stabilize that trajectory with an LTV feedback and compute the basin of attraction on it, OK.

So here we go. Pick a point at random, OK. And actually, I'm not-- I don't always play the RRT trick. So I could do lots of RRTs to try to get back to that point, but I'm actually going to just use this as my goal and do [? DR call ?] to get me there, to design a trajectory to get me there. If that works, that's perfectly fine.

So that's a trajectory. I didn't draw it nicely. It actually starts here, goes this way, wraps around, and comes to that red point, OK. From [? DR ?] call, it quickly designs that trajectory. Now let's back up and start computing the cost to go function, the Riccati equation, backwards to stabilize that trajectory. And as we go, we'll compute the basin of attraction of that controller, which has exactly-- I drew it in finite segments, but I hope you can see that's exactly the funnels, yeah?

If I start the system inside any of that blue region, and I execute the trajectory, the LQR, the trajectory stabilizer along that trajectory, it's going to take me around and get me to my goal and stay there, OK. So this is feedback motion planning happening. Now the cool thing is, I told you about the multi-query idea. I told you about all these ideas, talked about making very dense trees that handle all these situations. If you know the basins of attraction of your existing controller, you don't have to build a very dense tree.

I know, if I were to pick another random point that was already inside my blue region, I'm not going to get a lot of value out of adding nodes inside that blue region. So let's pick another random point, and if it's inside the blue region, we'll throw it away. If it's outside, I'll keep it, and I'll try to grow to it, OK.

So I get another random point, which is here. Going to pick the closest point in my current tree, which was just, in this case, a trajectory, connect that back. Now in this one, my dynamic distance metric, which was that LQR distance metric, connected and said this was the closest point. Looks a little surprising, but maybe the torque limits said that one couldn't get there, or maybe my distance metric just wasn't perfect.

But that's reasonable. It tries to go from here, add a little bit more torque, and drive out. And I stabilize that with the funnel, yeah?

OK, now I have two trajectories, and I've got a pretty good coverage of the space already. You can imagine I design a handful more trajectories, picking the state as I go, and I can really quickly and efficiently fill that state space with funnels which take me to the goal. Does that makes sense?

AUDIENCE: So when you do your multi-query, how do you choose which funnel you're in?

PROFESSOR: Awesome. Well, first, even-- so if I just want to execute this, yeah. And I want to get to that goal, it might be that I don't really have to do the-- so you could think of this as being every time being a multi-query thing. So every time I-- if I start, if I pick a point that isn't in any basin of attraction, then I'll try to connect and grow a tree there. If I, however, pick a point-- if it's execution time, I say the robot's got to run from here, I pick a point. It's already in the basin of attraction that I just execute that trajectory.

If, I think what you're alluding to is that it's in the basin of attraction of multiple points, then I pick the one with the lowest cost to go, because those are all estimates of the cost to go that are centered around that trajectory, OK. So for the simple pendulum, with damping and torque limits and everything set the way it was, this little randomized algorithm can fill the space with basins of attraction with just a handful of trajectories.

AUDIENCE: [INAUDIBLE]

PROFESSOR: I've never said it with so much--

[LAUGHTER]

--such dramatic force. OK? [CHUCKLES] It's the highlight of the class right here. OK, so this is exactly the feedback motion planning idea that I'm most excited about right now. Because we can suddenly-- for LQR controller, it depended on-- the thing we've worked out is, if the cost to go function is this, or in the time varying case, this, then I can come up with a very nice representation of the basin of attraction based on just a scalar value. And I could just start designing funnels through my state space. And the vision is, if you can think about the funnels as you build them, then you actually don't have to build too many trajectories to start filling the state space.

AUDIENCE: Why do you call it feedback motion planning.

PROFESSOR: Yeah. It's because I'm thinking about the feedback control, which is the funnel, as I'm doing the planning. Yeah. Do you agree why Koditschek's version is feedback motion planning, or do you not like that being feedback motion planning?

AUDIENCE: It makes sense. I guess I'm used to different funnels.

PROFESSOR: That's true. You are, yeah.

AUDIENCE: [CHUCKLES]

PROFESSOR: OK?

AUDIENCE: [? Think ?] [? so. ?]

PROFESSOR: So these are very much-- in Koditschek's case, there's no debate that each funnel is a feedback controller. I think of this as the same way. You could argue with it, because it's centered around trajectory design, which his is not. So this one has a little bit more of a feel of conventional motion planning. But by virtue of thinking about the feedback as I design the trajectories, it means I have to build less trajectories, yeah.

So it'd be nice to actually have the conversation about how these are related to float tubes. Mm-hmm. It's pretty similar in some ways. But these are very effective to compute the stable-- the basins of attraction, so I think it's relevant. Yeah.

AUDIENCE: Could you factor in actuator limits into the Lyapunov function?

PROFESSOR: Yes. So OK, actuator limits in the Lyapunov function are harder. So what you do is you-- everything is based on a Taylor expansion of the dynamics around the nominal. So a hard limit, if I linearize and I don't see that limit, then I'm not going to know about it. So there's a couple things you could try. And actually, I recommended to Mike earlier today that he should do this for his final project, is to do that, the case where the actuator limits.

So you could imagine making a soft limit, some sigmoidal limit, and having the gradients of that visible from your linearization point. Or you could imagine the LQR design that actually does both the quadratic cost and the bang bang synonymously. Haven't done it yet, but I think that's consistent.

OK, so I told you a lot about local trajectory optimizers. And today we said there were at least three good ways, I think, to make those trajectory optimizers into a more feedback plan. So the first idea was real time planning. And if it's fast enough, well, then we're all out of jobs, because we could just do that.

The second idea was building these trees and doing multi-query, keeping your tree around and just finding your way to the closest point of the tree every time you execute. And that has the nice feature that every time I execute, my tree gets a little bit bigger, and I know a little bit more about my robot and myself, yeah. And the last one was this feedback motion planning, which there are only a handful of ideas out there, I think, about feedback motion planning that people use. Koditschek's funnels are definitely the most prominent.

And actually, I think that the funnels should probably be on my list, but I haven't-- sorry, the float tubes should probably be more on my list, but I don't-- I've never made a strong enough connection. We should make that a goal for the rest of the class, yeah.

AUDIENCE: [INAUDIBLE] float tube or--

PROFESSOR: So there's definitely differences, but we should really figure it out. So [? Brian ?] [? Williams' ?] group does planning with float tubes that are, in spirit, similar to these funnels. Yeah. And so we should talk about whether you can design the float tubes for the class of systems that I care about in the class and stuff. Mm-hmm. Excellent.

OK, so you saw the email about the projects. If you have any questions about your projects, we could talk for a minute right now, or we could schedule a meeting before Thursday. There's a few ideas on the email we sent in the PDF that we sent out. If you're looking for more ideas, I've got a list of other ideas that I'm happy to share.

It's going to work best if you find a problem that you're passionate about, something that you got excited about in class or from your work, and you apply some idea from class. But the goal for Thursday is to say enough about it that I can give you some real feedback on your half-page write-up and try to help you with the scope and topic to make it a good project. OK? Let me know if there's any questions. See you Thursday.