MITOCW | Lecture 12 | MIT 6.832 Underactuated Robotics, Spring 2009

RUSS TEDRAKE: Today we're going to do sort of the second part of our discussion on walking. I want to start with what sounds like a little bit of a remedial question in some ways, in numerical analysis of these walking kids. Because it forces us to be more careful about talking about the dynamics through collisions and on Poincare maps. OK. So I actually want to start the discussion by just asking given we've defined a walking robot with some stance dynamics, and some sort of collision dynamics, like we did for the rimless wheel and the compass gait and the neat compass gait, quickly. I just want to ask the simple question. Can you find the stable limits cycles or even any periodic gaits of the system?

OK. So the first question, I want to find equilibrium, fixed points of the Poincare map. OK. So remember we talked about Poincare analysis of limited cycle behaviors last time. And we said that this continuous time, stable oscillation could be nicely characterized as a discrete time, fixed point analysis. If we're able to, for instance, for the van der Pol oscillator-- and call that x the next dot on the van der Pol oscillator. If we're able to design some surface of section, and just look at the state of the system every time it goes through that surface of section, then I can build a Poincare map.

On the Poincare, I've got a state x, which is just some potentially nonlinear mapping from the nth crossing of this map to the n plus 1 crossing of the mapping of the surface of section. So given I have some sort of bisecting surface here, what I'd like to do is now find the fixed points of that solution, which means xp star is just p of x p star. Simple enough. And that should tell me where I've got a fixed point, and, therefore, where I have a limit cycle. It doesn't actually imply necessarily that limit cycle is stable. If it was stable, then you could imagine finding the fixed point, for instance, of the van der Pol oscillator just by simulating the van der Pol oscillator for enough time. I'll eventually go and do the work for you and find its way to the stable fixed point if you just run along long enough.

But if it's unstable, that's not as easy to do. And for systems like the compass gait, it turns out the basin of attraction is sort of small enough that you could spend a long time trying to find initial conditions, which happened to simulate their way into the stable fixed point. OK. So you want to do better and get some better tools. Given what we've talked about in other parts of the class, if you want to define the roots of this-- the fixed points of the system, how would you do it? I just slipped a little bit and gave you half the answer there. How would you do it? I've got some big, complicated function p. I want to find the fixed points.

Close. So I want to find the roots of some function of xp. So it's equivalent to finding the roots-- the roots of pxp star minus xp star. So if I can evaluate p through simulation or whatever, then I can evaluate this. And I could just hand it to MATLAB or something and say, find me the roots of this nonlinear function. OK. And that's easy enough. We talked about the Newton method for finding roots quickly, and we talked about the constraint solvers that SNOPT and the like uses. OK. So in fact, we could just hand this to SNOPT as is, and ask it to do its thing. And it would do numerical derivatives and would find its way to some zero of that function.

We're going to be a lot better shape, though, if we can compute the gradients of that analytically. And hand it to SNOPT. It would take a lot fewer trials. And I want to sort of do that exercise quickly, computing the gradients of the Poincare map. Because it's going to be broadly useful in control. And because it forces us to really think about how we define that Poincare map. OK. So what we'd like to do, we just say it's easier if we have partial p partial x available for the optimization. OK. Just think ahead for a second here. How would you compute if I would just ask you on the street, I've got a Poincare map, how would I compute the gradients.

What would you think? What would you start trying to do? Yeah. So where we analyze the perturbations before? What kind of tools go with that idea? Yeah. So again around any given point which defines the trajectory, we could do backprop through time if we cared about a cost function. But remember the trick behind backprop through time, computing those gradients, the reason that was so efficient was because we were trying to find the gradient with respect to a scalar output. So backprop through time did a forward pass, and then connected it to that scalar cost function output. And then backed up a minimal state vector.

The forward version we talked about was just computing the gradients marching forward. And I called that real time recurrent learning, RTRL. And that's actually sort of 80% of what you need to compute the gradient here. Is there something I did wrong? No. OK. So RTRL, which we've talked about before and I'll say again real quickly here, is a lot of the answer. But there's a subtlety that I want you to see. OK. So let's be a little careful about how we go from xp of m over to xp n plus 1. That mapping is sort of a complicated-- there's a transition from this discrete time system to the continuous time dynamics, a simulation of those continuous time dynamics.

Potentially, an impact map at the end. We talked about the rimless wheel doing the Poincare map right at the impact. So let's do that carefully. So let me define xp at m as being equivalent to, in continuous time, x, the original vector-- I used a p to denote the discrete time, Poincare thing-- at the time of collision of the nth collision. In the walking case, we always did our Poincare map at the same time as the collision. And more generally, you could just think of this as sort of a collision with a surface of section. OK. In the walking case, because the collision has some dynamics, we want to be explicit about whether it's before or after those dynamics, before or after the collision.

So I'm going to call our discrete time state of our system to be the state of the system immediately after I process that collision dynamics. So I'm going to put a plus there. Minus would be just before the collision. Yeah. They're both at the same time, but there's an instantaneous dynamics. OK. So I have this. I know how to go from xp to a continuous time thing. And then I know how to go from this to march that forward. If I know when the next collision is, then I know that had better just be x of tc plus n plus the integral from tc plus n to tc minus n plus 1. Let's just say my dynamics are x dot equals f of x. I forget about control for a second just to keep it simpler.

Once I go from my discrete time thing to a continuous time thing, I just integrate my equations of motion forward until I collide with the surface of section again. And I want tc defined somehow. I want to somehow define that surface of section. And to define that surface of section and that collision time, I'm going to define-- or more generally a collision-- as some function that equals zero. There's some manifold of states that I can describe by this function, which will define my surface of section. So what was it for the van der Pol oscillator? I designed my surface of section to be this thing going forward. So what the surface of section is defined as some function which just happens to be zero here.

So a perfectly good candidate would be like the distance, the signed distance let's say, between your current state and that manifold. You OK with that? I don't think I said that very well. But if I want to know-- define my surface of section, just define a function of state, which is non-zero everywhere and zero where the surface I care about is. Yeah. And that's going to allow me, as you'll see, to do things like taking the gradient of knowing when I'm going to collide with that surface of section. So this could just be for instance the signed distance between, let's say, the foot of the robot and the ground. Yeah. So when the robot's foot hits the ground, that thing's going to be zero.

When it's below the ground, it'll be some negative distance. When it's above the ground, it'll be some positive distance. OK now that's going to allow me then to define this. At a time of collision, this thing is going to add better be zero. Good so I would I'd want to define it in a way that doesn't include that. So I could put a-- so the robot's foot colliding with the ground doesn't have that problem, if you think about it. But I do typically put something like a velocity term in it. Let me just write down what I would use for the rimless wheel. Actually, I've got it here. So I do phi of x is just sine, the sine function of theta dot times theta minus gamma minus alpha. So if you don't remember all those symbols, that's almost useless.

But this is saying-- the sine of theta dot tells me if the robot's moving forward, then I'm going to look for the collision with the ground at this angle. And if the robot's looking backwards, I'm going to actually look at the collision with the ground at that angle. And that's how I end up getting the surface of section which doesn't go-- just goes in the top half and bottom half of the thing. So you can divide whatever function, but you just want it to be not 0 here. OK we're getting closer to having this be well defined. In fact, this is a very common way to define a collision in any hybrid system.

So systems that are continuous with discrete impacts are called hybrid systems. If you wanted to simulate a hybrid system in MATLAB with ODE, then you would very much do that by defining a function, which they call a collision function or a collision event function, which has got 0's here. And you can hand, as your options to the ODE solvers in MATLAB, event functions like this. And they will very carefully watch a crossing of that event function across 0, and pinpoint the time and state at which that thing goes across 0. So this is a very common notation in hybrid systems kind of worlds. It turns out if we do care about finding this gradient of the Poincare map, then this is going to be also be the enabling thing which allows us to compute that gradient.

OK So we have this x of t minus. We've gone from just after a collision. We've integrated forward over a continuous dynamics. Yeah. So now we're at the time of the new collision, tc minus n plus 1. We're going to compute the new dynamics by having some collision function. In the rimless wheel, it was just the thing that took out some energy because of the inelastic collision with the ground. So the final piece of the puzzle here is x of p n plus 1, which is equivalent to x of tc plus n plus 1, which is some impact dynamics. I'll just call it big f of x of tc minus n plus 1. In 90% of cases that I would care about, it's only a function of x.

I like to write in the more general form in case you have things like moving obstacles. Let's say. In the surface of section case for limit cycles, I think it'd be pretty unusual to have a direct dependence on time. But it doesn't make an our derivation any more complicated than if you had-- you wanted to worry about collisions with moving things or something like that, you could do that. OK. So this is a sufficient recipe now for simulating the Poincare map. If I want to just evaluate in the simulation with the Poincare map, someone says I'm at state xp on the nth Poincare map, I'm going to turn that into my continuous time thing. I'm going to integrate forward my dynamics, just like we always do. But I'm going to stop at a particular time, tc minus, which is defined as the time when it causes phi to equals zero.

And then at that time, in the case of the walking robots or any collisions, physical collisions, you might have to implement the discrete collision dynamics. And that takes you back to the next map. Good. So how do we take the gradients of that? It's almost trivial, almost trivial. We know how to take the gradients off of this. We'll do it again, just in one line. But we know how to just integrate forward the gradient dynamics to compute the long term gradients of x. The only subtlety is that-- so when I'm computing the gradient, I'm changing xp a little bit. I'm trying to do a sensitivity analysis between x. If I change xp a little bit, how is it going to change xp n plus 1?

So the only subtlety is that when I change xp a little bit, it can change my collision time. So you have to make darn sure when you're taking the gradients, that you capture the changes in collision time that are due to your change in the initial conditions. Other than that it's almost exactly the RTRL code that we've done before. And that turns out to be not so bad, so I want to do it. All right. So we can just go forward with the chain rule here, so dxp of n plus 1 dxp of n. Well that's going to be, first of all, the gradient of this f, partial capital f partial x and then times the gradient of the inside. The inside is defined by x at a certain time. So this is where that comes in.

OK. So first of all, let me say we're doing we're doing this linearization around some nominal trajectory. We have some xp0 that we're linearizing around. That implies we have an xt trajectory that we're linearizing, around which I'm calling x0. And it implies that there's some nominal time of impact-- time of collision that we're linearizing around, which I'm calling tc0. OK. So the obvious gradient here, partial f partial x, gets us past-- gets up backwards with respect to the impact dynamics. And then we have to figure out what the state of x is relative to the initial conditions, at time evaluated at the original impact time.

And then the subtlety is that I can also increment or decrement the index of the integral, the limits of the integral, which has the effect of adding this term again modified by that increment in the limit of the integral. So I get f of x times that increment. OK. So let's do this in a picture. So let's say I have some thing, some manifold defined by phi equals 0. And I have some nominal trajectory that I took to get there. I got to my switching surface. I'm calling this x0 of t. And this one here is x0 of tc0 plus n, to be entirely confusing. It just takes a lot to write. I could write that equally well as xp0 of m.

And what I'm trying to figure out is what's the state going to be if I'm going to simulate my system from slightly different initial conditions when it hits the map. So using my multi-colored approach here, if I take some increment in x0 and I simulate my new dynamics forward, then maybe I get something pretty similar. But at time-- what my original end of time, if I evaluated this integral for the same amount of time, there's no reason to expect I would get back to that switching surface. So this is my modified thing at tc0. But what I would figure out is where it's going to be the next time it hits to the surface. So you can do that by figuring out what this difference is. This is partial x of t given the initial conditions at time tc0. That's this term here, corresponds to this. Yeah. But I'd better also add in the dynamics of this thing pushed forward by the amount of time necessary to get me back to the surface for that collision. Yeah.

STUDENT: So you never take the derivative of the p function?

RUSS TEDRAKE: You do. I'll show you. So in order to get this, we're going have to take a derivative of the p function. I can't tell if people are bored by this, or intrigued by this, or don't care. OK. Does that makes sense? Good. All right. So how do we figure out the change-- the increment in time, given the initial conditions? Well it's defined based on that phi equaling 0 is the thing that defines this collision surface. So it's going to be used in computing that increment in time. OK. So if phi equals 0, then it better also be the case that dphi dxp of n-- and let me just write this-- tc minus tc n plus 1, x of tc minus n plus 1. This is just the derivative of that thing based on the changes of in xp.

I've defined that. I'm saying that even though I changed xp, phi had better still be 0. I'm integrating until I hit that surface. So this thing a better equal 0. And I can take that derivative with partial phi tx, partial x-- t evaluated those same places. OK. I'll save myself from writing that last equation, but obviously I can now solve this for dtc dxp. Yeah. Stepping back just a second, I'm making an increment here. I'm defining that, even though I made that increment, I still better get to phi equals 0. I can look at the change in phi that could potentially occur. It's going to also depend on the change in x and the change in final time.

And that allows me to solve for the final time that must have made that happen. That must have made it so I get back to the switching surface. Yeah. OK. That's really the only thing you need to know to make all of your tools we've used for loop optimal control, for instance, for the acrobat and the cart-pole work for the walking systems. The additional advantage of thinking about this now as on the Poincare map is in some ways we could do even easier control.

OK. So what did I just solve for? I solved for partial p partial x. Absolutely. Please. Yeah. Good I want questions. No, not necessarily. I mean this to say, so it happens-- it happens that oftentimes at the surface of section we want to compute a discrete collision dynamics, which is just some other function which I'm calling capital f. I don't mean that to be directly related to that. I can see that coming right after the integral that's confusing. Want to I call that something different? We can call it something different. Yeah. This is the equation-- well in the rimless wheel example, it actually says I'm going to change coordinate systems back to where my new leg is on the ground.

So I'm actually going to do a discrete change in theta, and I'm going to take away some of the theta dot. Because I've lost some energy into the ground And in general, when you have collisions in mechanical systems, if you model them as impulses, you're going to have some function like f. All right. This is equivalent, right, to being the partial p partial x, Everybody sees that? OK. Not a little bit of change of phi. What we're changing, the thing we're changing, is xp.

STUDENT: And x0 and all the nots are basically the initial trajectory we would have taken without changing.

RUSS TEDRAKE: Yes, because we're doing everything as an increment. We're doing an incremental analysis. So I want to say that the new tc is going to be tc0 plus this increment. I've defined the problem that way, again. So even if I make a small change in xp, I still want it to be that a minute collision point-- I'm not going to call it a collision point until phi equals 0. And if phi's going to equal 0 along everything, then it certainly equals 0 on an increment in xp. Yeah. So it's tempting to say-- so we've got gradient based calculations flying around here. We've been doing a lot with that. I think they're pretty powerful.

It's tempting to say that if I have a discontinuity, then it breaks all my gradient calculations it doesn't break them, you just have to be more careful about them. Yeah. OK. So we can take a gradient through the impact of a walking robot. No problem. You just have to use this. OK. This will be the same if you're doing a ping pong robot or something like that. Anything with collisions, a mobile robot that runs into a museum visitors or something like this, would also have an impact map. OK. So that's all you need to use all those methods we did before.

OK. So if I wanted to now optimize the trajectory, let's say, of my periodic system-- if I had my dynamics and now we're back. I did this just in the simple case of f of x, but if I had f of x, u and I wanted to optimize u in order to do something good on the Poincare map that minimizes the cost function, I can take the gradients. I can do my optimization. But this idea of actually changing to a discrete time system is a bit empowering too. You can always do it. It's not that you only do it for walking robots. Any continuous time system, instead of looking at it at every t, I could look at it at discrete intervals of time. I could take a ball that I throw across the room, has no impact whatsoever well at the beginning, and then I could look at the system at x of time 0, then x at t equals 1 second, x of t equals 2 second.

And I could build a discrete time system for any of these continuous time systems. It's particularly nice to do it on these walking robots, because, well, let's do it. So what have I just done? I've computed partial p partial x, which is telling me that I'm going to have-- I'm approximating, it's a Taylor expansion, of my dynamics around some nominal point. If x nominal is a fixed point, or if I just change my coordinate systems to x bar is xn minus x nominal n . And I could do the time varying thing if I want, or I could just do it simply if xbar0 is a fixed point, then I'm left with about a model like this.

Careful here. p's everywhere. p n plus 1 is a xbar p of n. OK. I can immediately look for the stability of that model by taking the eigenvectors, eigenvalues. Stability conditions are discrete time so those eigenvalues had better be bounded by 1 and negative 1. It's not the same condition but perfectly easy to analyze the stability of the fixed points. So now for the-- if I just cared about analyzing the passive walker, I've got ways now by computing partial p partial x, handing it to SNOPT or something to solve to find the roots. I can find the fixed points and it also happens that by computing partial p partial x I can evaluate the stability of those fixed points.

Yeah. The rimless wheels fixed points, if you remember, there was two of them. One was a rolling fixed point. One was a standing still fixed point. Both of them locally stable. The compass gait, if you remember the compass gait, It also can have multiple fixed points. It has one fixed point for a nominal slope. It has one fixed point that's walking, and it actually has an unstable fixed point. Now the cool thing is-- so Ambarish Goswami, who's a friend and does a lot of the initial confiscated analysis. If you start inclining the ramp steeper and steeper, then those fixed points change as a function of the dynamics. Something really interesting happens. It's not surprising that these are complicated dynamics. But you actually at some point, at some critical angle, you get a period doubling bifurcation.

So you get an extra point that corresponds not to a one step fixed point, but to a two step fixed point. So what does that mean? What does that correspond to physically? What is a two step? What do I mean by two step on the Poincare map? So at x of n plus 2 is going to equal x of n. But x of n plus 1 doesn't necessarily. But every other thing is going to be the same. So what does that physically correspond to in the walking? It's more of a limp I'd say. Yeah it's some asymmetric gait. It's like this. These robots are doing potentially asymmetric thing. Every other footstep ends up landing in the same place but not every footstep. OK. So it's actually-- there's a lot of interesting work just looking at these passive models and just looking at the stability of these passive gaits. OK.

I do want to say make one note here, If you're looking at the eigenvalues of A, the way I've defined it. So the trivial condition here is that the eigenvalues of A had better be less than-- the magnitude had better be less than 1. There's different notations for working on the Poincare maps. My notation is to denote x of p where x is the same size as the original vector. Because that's more useful in simulation that's more useful most of our computations. But remember the Poincare map effectively reduces dimensionality by 1 And in the way I've written it here, if you just look at A that comes out of a stable periodic oscillation, there's actually, the way I've done it, there's always a trivial eigenvalue of 1, which doesn't degrade the stability of the system.

What does that eigenvalue correspond to? If I take my stable rimless wheel, I compute A. I take the eigenvalues. I see that the system stable. It goes to my nominal thing. This is the standard stability criteria for discrete time system. But I notice that there's actually always a trivial-- there's one eigenvalue that equals 1. Good. People see that? There's one direction that I can push it in which it does not reject that disturbance, and that's the direction-- if you push it along the limit cycle. So there's two ways to do it. I'm sorry to make it complicated, but there's two ways to do it. If I were to have redefined my coordinate system on the map, I wouldn't want to call it x anymore, but something that reduce the dimensionality of the map by 1. And then I could use this condition alone.

But in the way I've done it in the notes and in life, I like to keep this sort of as the same dimension of x. And then because I want limit cycle stability, it's absolutely the case that one of the eigenvalue-- there is a direction in which those disturbances are not rejected. That's what I want. Yeah. OK. So the way I describe it in the notes is that what you want for stability of the periodic gait is you want to ignore the first eigenvalue of one. The rest of them better be-- I want it to be only one direction that doesn't reject disturbances. You with me? OK. Then there's some other direction that doesn't reject disturbances and then I start questioning whether it's stable. I wouldn't call it stable in the sense of a limit cycle stability.

If I could push in some other direction in state space and doesn't get rejected, then it's not going to return to that orbit. There's a special direction along the orbit, which I'm allowed to push, and that's what defines limits of stability. But there's only one direction. Even in high dimensional space, a 50 dimensional robot going around, there's only one place that trajectory is going forward that I'm allowed to push and not reject the disturbance. Every other one better converge for me to call it stable. OK. We did the case for the non-controlled system. But we're only a stone's throw away from doing the actuated version. I'm not going to write it out again, but let's say-- let me do it a little carefully just to say what I mean carefully.

So if I want to do the controlled case, get back to a system like this. We talked about in the policy gradient-- in the policy research world-- that I could then define u to be a tape of u's, or come out of a linear feedback control or whatever sort of parameterization. But I like to think of it as being some function which depends on a parameter vector, alpha, and can generally depend on x and u. So there's a different way to think about control in the discrete time Poincare sense. One way to think about it is, let's say every time I hit a surface of section, or my foot hits the ground in the walking case, why don't I change the parameter vector alpha OK. But I'm going to make decisions only once per step. And then I'm going to execute this policy for the duration of that cycle.

And then the next time my foot hits the ground I'll make a different decision about alpha. You can imagine. So let's say that pi of alpha and my compass gait walker, we'll use this example in simulation in a second. Let's say I have a controller that runs during the limit cycle which tries to set my interleg angle to be some desired value. Every step I take the only decision I make is what should that desired interleg angle be for my compass gait. But over the course of the cycle, I'll simulate a PD controller that tries to make that interleg angle happen. But my discrete time decisions are just what interleg angle should I be. Fumi has got a compass gait that works with open loop trajectories that play out, where his major parameters are the frequency and phase of the periodic input. It's amazingly stable coming out. It's like a beautiful example of open loop stability. OK.

So Fumi and I have been talking about changing the parameters of his open loop controller once per step, in order to try to regulate the behavior of the system. OK. There's lots of ways you could do it. But, generally, what that gives you-- if you just try to compute these exact same gradients again, we've computed the gradient with respect to the initial state. But we could also compute a gradient with respect to alpha. And what that could give me here is a model like this. I call it B because I'm thinking of alpha like a control decision. Where alpha bar of n is the difference between my control decision at n minus whatever the nominal alpha is. OK. So let's say I find a stable limit cycle where I just command the interleg angle to be, I don't know, pi/4 every step.

And I can find a stable limit cycle behavior. Well if someone pushes my robot, computing these A and B matrices give me a nice, simple way to make discrete decisions to try to stabilize that robot. If I'm away from my limit cycle, I can correct for those differences by just saying, OK, next time take a bigger step and then take a smaller step. And how would I design that rule to change alpha? How would you design it if you wanted to make a feedback law an alpha here? This is a discrete time linear system. I could just hand it to MATLAB and call LQR, discrete time LQR. Give it A, B, Q, and R. It'll give me back a negative k alpha, a kx to find alpha.

Are you with me on that? So the discrete time summarization of the dynamics in this way can actually make it very natural and very simple to compute controllers, which stabilize a walking cycle for instance, where you make decisions once per step. I want to show you some work that Katie Byl did in my lab, where she did it on the nonlinear case for the compass gait. Instead of linearizing this, she went right with the full nonlinear Poincare map. She just did x of n plus 1 is some Poincare map of x of n parameterized by alpha, or alpha n. And the compass gait, how many dimensions? What it is it's four basic dimensions. There's two angles and two velocities. And it's just about the right size that you can try discretizing and doing value iteration.

So Katie had some nice work showing that on the compass gait we can really get a pretty good sense of the nonlinear optimal solution by just popping this discrete time control problem into a value iteration, doing some work, and computing back an optical feedback policy, which tells you where you should step every time in order to stabilize your gait. One better, if you add one more dimension to your compass gait robot, you can actually think about where it is on terrain, and make it walk over rough terrain. So maybe I didn't spend enough time in these two lectures convincing you that nobody knows how to make robots walk on rough terrain, but-- oh shoot, it's going to take a second here. But you might know for yourself that you don't see a lot of walking robots walking on rough terrain yet.

Big Dog's sort of the exception. Big Dog seems to be pretty good. We've got a little dog upstairs in our lab, which is also pretty good I think. They're different. Little dog got can see the terrain and Big Dog can't, so we're supposed to be doing the long term research for Big Dog's idea. Now we've had that entry music here. It's a very dramatic simulation that follows. OK. Here's Katie's work on compass gaits on rough terrain. A little compass gait robot. Its making control decisions every footstep. It knows where the terrain is. Pop it into to a big value iteration solver. And this thing can walk over almost anything, It applies to talk at the hip width the PD controller I described, and it also puts in an impulse at the foot, one time every post impact. So it can push itself forward a little bit.

So this thing it's pretty good. And it's just converting this sort of complicated looking problem into a valuation problem. It doesn't think about the slope of the terrain. It could. But the reason it's tractable, actually, if you can tell by the footprints, it wraps around. So we've got a limited state space in x. We ignore it. If you look real carefully, I think the leg gets a little shorter every time it goes. But the mass looks like there's no dynamics like that. We built the robot the same way to have a little tweak toe that pulls up. It's trivial in simulation. Yeah. How do we do it in real life? Not as well as we'd like. Fumi has got a design that's got a servo that pushes as fast as we can. We've talked about pneumatics, that would be a little faster.

But we haven't run that, run them yet. It's interesting. So she actually did the case of just hip, just foot. They both work. They both stabilize the walking on moderate terrain. But together they're much more stable than the other. The thing with the hip one, it always gets in these configurations like this and falls backwards. You need a little bit of foot energy to get over. And the one that pushes off of the toe keeps putting its foot down in the wrong place. There's one thing that Katie's simulations had problems with. You guys in my lab know. But there's a pretty sort of very visual-- I should find the video maybe.

But there's a very visual thing that we had problems. What was the problem I always talked about with valuation iteration? Even if you could pile the space, you got to watch out for something. Yeah. Your discretization, if there's hard discontinuities, you can do things wrong in your discretization. Your discrimination can be a poor approximation of your continuous thing. So Katie stimulated a bunch of terrain that had holes in it like cliffs. And had these beautiful policies that would choose their step across Karate Kid sort of style terrain. But every once in a while, the mesh points would land in the wrong place and the stupid thing would put its foot right in the middle of the hole, right down into the ether.

This is sort of the textbook case of value iteration resolution problems. OK. So walking isn't any harder than any other robot really. There's mechanical challenges. You have to carry your actuators. You're typically dealing with actuator saturations a lot. So it's definitely underactuated. I don't mean to say it's easy, because I think the acrobats are rich and hard. But it's not significantly harder than the acrobat and this is, remember, the acrobat dynamics. The only two differences are that you have to worry about-- you tend to think about Poincare maps to define stability, and you have to worry about these collision dynamics. You get 100 degree-- I'm not showing you 100 degree of freedom robot here walking along but you could.

I mean we're getting there. So the dork hall type methods that we're using with the LTV linearization. We think that's going to work pretty nicely for a little dog and that's 36 dimensional, walking on very rough terrain. So there's been a lot of work in the control of this, sort of outside the optimal control view of the world. I'd be doing you a disservice if I didn't tell you a little bit of it. Make sure I said everything I wanted to say. There was one other point I wanted to make. There's something you can do in discrete time that you can't really do in continuous time when it comes to stabilizing controllers. Anybody get it from just that? I said it very obtusely, but yeah

Yeah. Good. So in discrete time, I can actually find an action, potentially, depends on B. I could find an action which would actually drive me to zero, in my aerodynamics, in a single step. In continuous time, getting there arbitrarily fast means setting your gains arbitrarily high. In discrete time, that's not the case. I can set a finite magnitude gain that will get me to zero in a single step. So that's called deadbeat control. It's a very beautiful goal to have for a walking robot. if I get perturbed by the terrain or by someone hitting me with a baseball bat or whatever-- you should see the videos that the robotics people make of walking robot-- then a very beautiful goal is to say that before my next foot hits the ground, I'm going to cancel out all the error for my disturbance.

Certainly you can do that here if you have no limits on alpha and if B is full row rank. Then it's algebraic equation to compute what alpha n had better be. I could just say, if you want something that could basically look like feedback linearization, I could just cancel that out. Sometimes you can't do it, but I want you to know that's a beautiful goal for control. We'll see in the running bit, we'll see some running models where you can do that B control. The compass gait, if you had the right parameterization, you could do compass gait control. John and I were debating this last night. I don't think that the PD controller is probably going to be enough to give you that B control. Because it won't simultaneously take out your energy and get your foot in the right place.

But it's possible. You could actually do the analysis and answer that question. OK. So thinking about these things as discrete time control problems is pretty beautiful. More generally, though, we can do control through the swing phase, in this case. Or between the surfaces of section in a general limited cycle case. And if you want to do that, you could do you could do a shooting method. You could do deer call. Optimize some trajectory. You have to be careful because the time can change during your optimization, the duration of your trajectory, but you could do that. And then optimize it. But just like we showed in the acrobat and cart-pole case, people have come up with more-- I don't want to say problem specific-- but more sort of problem specific solutions.

So Goswami, Ambarish Goswami did some nice work. Can anybody guess? What's one of the dominant nonlinear control ideas we talked about for the acrobat and cart-pole. PFL, but that led to energy shaping. So Goswami did a nice controller based on energy shaping. And he showed that you take your nominal confiscate with its fragile little basin of attraction, which he computed by sampling. Pushing the system until it fell down. And he showed that with an energy shaping controller, which is derived exactly the same way we derive the other one, but I can regulate the energy of the system to put me back on the energetic orbit. Because this thing is with zero torque. It's passive in the swing phase. So I can just drive myself up to the place where I will passively fall down and hit the right place on my Poincare map. And that worked locally. That's a good idea.

There's another bunch of work that's become popular in our world. Eric Westervelt and Jesse Grizzelle and these guys they talk about hybrid zero dynamics. And more recently, our friend who's going to join the lab, the advisor in Manchester. These guys have been doing sort of similar work using more optimal control derivations and similar hybrid zero dynamic kind of methods. Let me just cartoon the idea. It's sort of similar to the PFL idea. But these hybrid zero dynamic methods are one of the places where we actually have proofs of convergence. And the way they do it, like in partial feedback linearization, is that when the system is-- let me say it carefully here. They find some desired trajectory of the robot, which has a periodic gait that they like. It has actuated joints that follow this trajectory and passive joints that follow this trajectory.

If you do a collocated PFL, if you just regulate the dynamics of your system, you're actuated joints to follow that trajectory, then the passive one pretty much has to do the same, has to do the right thing. OK. So the results that these guys on the hybrid zero dynamics-- they talk at the zero dynamics of the resulting output dynamics you drive to drive to zero. Yeah. When your PFL controller has done its job-- and has it doesn't have to be a PFL. It could be a speed controller. When you've driven all your actuated joints to the desired trajectory, if you parameterized that trajectory off let's say the ankle angle, then your whole big complicated bipedal robot, five link ten link whatever. Looks like a remote wheel rolling around the stance foot. And that's a beautiful idea. So maybe the bigger idea here is actually.

A lot of the walking robots you can think of as having a lot of actuators at all the joints and just having one passive joint. So a pretty good idea for a walking robot is to just drive all of the actuated joints as a function of the passive joint. And that gives you a dynamical system in one variable. It's a more complicated dynamical system than the rimless wheel. It's one that you can design but the analysis reverts to basically the rimless wheel type analysis. OK. So this hybrid zero dynamics is roughly that idea. And then you can sort of do a rimless wheel. I have had this debate with Eric Westervelt many times. Yeah. I think it is. Yeah. That's a good idea. Just like PFL is great. We should use it for a little while until we figure out something better to do. I think they'd admit that too.

Their stability guarantees are when you've driven the aerodynamics to zero. You squash the aerodynamics. And you have to have gains that are high enough to squash those arbitrarily fast. Now the bigger idea I think is that I mean you can do sort of softer feedback, and get some of the same type of performance, but the theory does depend on squashing. So all this stuff I talked about I think is sort of the right way to think about walking control. It's a small fraction of what people actually do in the walking robot world. By far, the dominant approach is these Honda ASIMO type robots. Anybody see HRP4C yesterday. What does it look like? It looks like a model. They actually videotaped a Japanese model walking in high heels, and they built one of their newest that's the AIST robot by Kawada Industries.

It's got now sort of a female Android head on it. And it works. I mean, I just saw a few videos. But it looks pretty good. The shapes are more feminine and the gait was a little bit looser. But most of the walking robots out there today, most of the successful ones, are all pretty similar to ASIMO. And ASIMO doesn't do any of this stuff really. ASIMO plays one trick. I can say it is sort of in a single line. They try to keep one foot flat on the ground, and assume that foot is bolted to the ground. And they pretend they're fully actuated system and do their trajectory tracking. And when they're doing their trajectory tracking, they have to make darn sure that foot doesn't roll off the ground. But if you watch the videos of ASIMO, you're going to see it's always got one foot flat on the ground.

Running is a small excursion. They just give up on stability for a little bit, until they go back to the place where they can catch on the ground. OK. That's relatively unappealing compared to these I think, because it uses a ton of energy. I told you in the first lecture it uses 20 times as much energy as a human when it walks. People have used these type of methods to make a robot that uses the same kind of energy economy as a human. And it's just walking with its foot flat on the ground. It doesn't work on rough terrain It's not walking as fast as we are. All the energetics of heel strike, you have to do all that with active control because you can't sort of reduce your collision if you're going to land with a flat foot, so this is a better way to do things. Excellent That's a quick preview into the world of walking. And really the key message here is it only takes one or two more tools to turn it back into the acrobat problem. OK. Midterm on Thursday. If you have questions about that, ask John. Ask me. If I do disappear, I apologize. But it's for a good reason. And we'll have a good spring break. And I'll see you soon.