Underactuated Robotics

MIT 6.832 | Spring 2019

LECTURE 10 Trajectory Optimization

Big Picture up til now:

Dynamic Programming
- General: Linear/nonlinear but with additive cost
- Tried mesh (low dimension) ← discretize over space which is hard to put bounds on error
- Function approximation? (e.g. reinforcement learning)

LQR:
- Scales very well
- Linearization means Soln only valid for some ROA

Lyapunov via SDP/SOS
- Scales well (10+ dimensions)
- Can find soln for all x $\forall x$
- SO far just used for stability analysis rather than synthesis Q!
  - e.g. Found ROA for ~~pend~~ using LQR by using SOS w/ S Procedure to find/maximize region where $\dot{V}(x) < 0$ Choosing $V(x) = x^T S x$
    ↑ cost-to-go from LQR as good guess
- Con: is limited to 'simple' control fns (work for polynomial / variable transforms e.g. sin/cos) but can't capture complex controls that come out of TO
- Con: Doesn't know a way to use this to synthesis stabilizing control of inv. pendulum

Other Extreme
- Solve for 1 IC to combat high dimensions rather than global control
- tend to grow linearly with dimensions rather than exponentially

- Basic Trajectory Optimization:
  Formulation

$$\dot{x} = f(x,u) \quad \min_u \int_{t_0}^{t_f} g(x,u)\,dt$$

s.t. $x(0) = x_0$
Constraints

- Discrete time case

$$x[n+1] = Ax[n] + Bu[n]$$
$$\min_u \sum_{n=0}^{N} g(x[n], u[n])$$
s.t. $x[0] = x_0$ ...

↑ Can't pass directly to solver since x not defined explicitly

$$g(x) = \sum_{n=0}^{N} x^T Q x + u^T R u$$

- note: w/o constraints same as LQR
- (QP) can still add linear constraints
- QPs can run at 1000s Hz
- Many variations that remain convex (fast w/ global min)
  - Eg min time soln for convex opt
    Solve w/ constraint
    $x[N] = x_G$ & line search
    by keep reducing N

→ Transcriptions to numerical implementation:

1) Direct Transcription ← solve all vars at once
   add x as extra decision variables (slack vars)
   as a fn of u

2) Direct Shooting Method ← Shooting b/c solve sequentially in time
   Solve $x[n]$ as fn of $x_0, u[0] \cdots u[n-1]$
   - Still linear constraint
   - Fewer decision variables but can be less sparse problem
   - more difficult to add constraints
   - Produce uneven gradients (u_1 high sensitivity)

- Debate which is better for nonlinear systems
  Russ prefers direct transcription

- Back to Continuous time

$$\dot{x} = Ax + Bu$$

$$x[n+1] = x[n] + \Delta t\,[Ax[n] + Bu[n]]$$

↑ Forward Euler or Runge Kutta ...

- Better than mesh approach from dynamic programming b/c easier to quantify error

- Sidenote: For unconstrained Linear system → LQR
  Constrained " → MPC

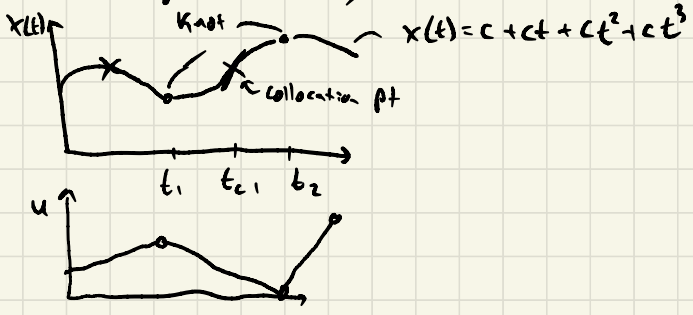
- Chose not to use non-linear dynamic MPC b/c risk of not finding a soln

- Sweet spot # params vs numerical integration accuracy

↳ Direct Collocation

  $x(t)$ as cubic spline
  $u(t)$ 1st order hold

- 3rd order accuracy



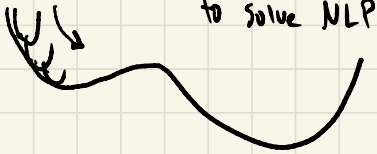
- Collocation pts - Constr slope

$$\dot{x}(t_c) = f(x, u)$$

- Note: Can shrink/grow Δt but it creates non-linear optimization to make them decision vars

- Sidenote: PyDrake calls auto detects linear vs nonlinear & calls diff opt algos

- e.g. SNOPT SQP - sequence of QP to solve NLP



- Sidenote: Can use traj opt to solve IK of robot end-effectors

- intuition

non-homotopy const → use convex optimization

LECTURE 11 Trajectory Stabilization

- Linearize along a trajectory

e.g. pendulum

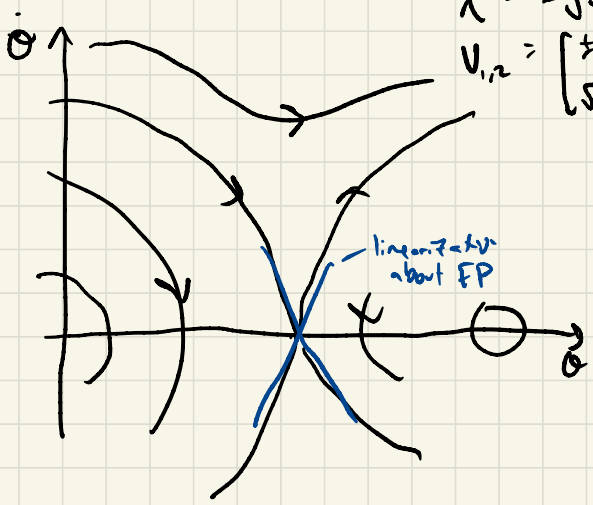
$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ \frac{1}{ml^2}\left[u - b\dot{\theta} - mgl\sin\theta\right] \end{bmatrix} \xrightarrow{\substack{m=l=0 \\ b=0 \\ g=10}} \begin{bmatrix} \dot{\theta} \\ u - 10\sin\theta \end{bmatrix}$$

$$\dot{x} \approx f(x_0,u_0) + \frac{\partial f}{\partial x}\Big|_{x_0,u_0}(x-x_0) + \frac{\partial f}{\partial u}\Big|_{x_0,u_0}(u-u_0)$$

$$\begin{bmatrix} \dot{\theta}_0 \\ u_0 - 10\sin\theta_0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ -10\cos\theta_0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\lambda = \pm\sqrt{-10\cos\theta_0}$$

$$v_{1,2} = \begin{bmatrix} \pm 1 \\ \sqrt{-10\cos\theta_0} \end{bmatrix}$$



← linearization only good @ fixed pts in this coord system (given smooth system)

- Moving Coord System

$x_0(t)$ $u_0(t)$ (e.g. from traj opt)
Consistent w/ dynamics

↳ $\bar{x} = x - x_0(t)$ $\bar{u} = u - u_0(t)$

$\therefore \dot{x} \approx f(x_0, u_0) + \dots$
$\dot{x}_0(t)$

$\dot{\bar{x}} = \underbrace{\frac{\partial f}{\partial x}}_{A(t)} \bar{x} + \underbrace{\frac{\partial f}{\partial u}}_{B(t)} \bar{u}$ $\quad \dot{\bar{x}} = A(t)\bar{x} + B(t)\bar{u}$

↑ Powerful covers a lot of control. Eg LQR works

$\bar{x} \to 0$ then $x \to x_0(t)$

- Time Varying LQR

$$\min_{\bar{u}} \int_0^{t_f} \bar{x} Q \bar{x} + \bar{u} R \bar{u} \, dt + \bar{x}^T(t_f) Q_f \bar{x}(t_f)$$

← may not be ∞ if for TO output

Can also make these $f_n(t)$

$J(\bar{x}, t) = \bar{x}^T S(t) \bar{x}$ $\quad \dot{\bar{x}} S(t) \bar{x}$

$S(t) > 0$

$$0 = \min_u \left[g(\bar{x}\bar{u}) + \frac{dJ}{d\bar{x}} f(\bar{x}, \bar{u}) + \frac{dJ}{dt} \right]$$

$\bar{u} = -R^{-1} B(t) S(t) \bar{x}$ ← similar to before
$= -K(t)\bar{x}$

← Boundary value problem defined at t_f so solve it backwards

$$-\dot{S}(t) = S(t)A(t) + A(t)S(t) - S(t)B(t)R^{-1}B(t)S(t) + Q$$

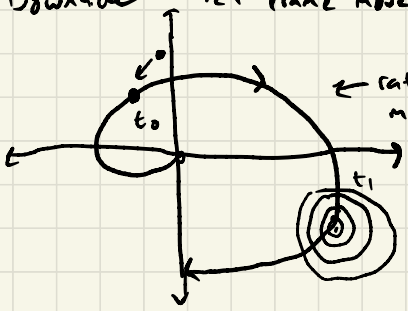
↑ Differential Riccati Equation

1) $S(t_f) = Q_f$
2) $S(t_f) = S$ ← final cost-to-go once reach fixed point is cost-to-go of ∞ t LQR

↑ LQR about vertical fixed pnt

↑ smart way to choose final cost

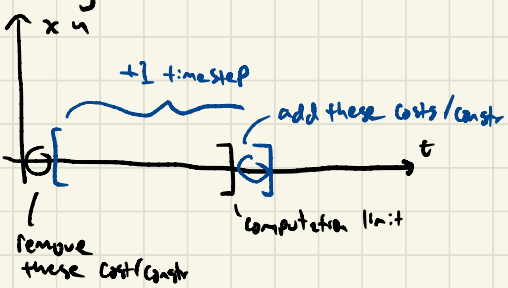
- Downside: ref frame moves forward regardless of where we are



← rather than project to closest point we will move back here before moving forward (unlike MPC which would reinit to x_0 each time)

← $S(t_i)$ quadratic cost fn moves along traj w/ origin along path

- Unconstrained Linear → LQR
- Linear Constraints → Linear MPC
- TVLQR - Locally stabilizes x_0 u_0
- LMPC - locally stabilize constrained system
- Cant solve QP for $N \to \infty$
  - need finite horizon
  - Receding horizon control



- Guard against infeasibility
- Cost @ end < cost removed in beginning ← Can get this by adding terminal constraint $x[N] = x_G$

↳ $V(x,k) = \min_u \sum_{k}^{k+N} x^T Q x$ st ... ↑ guarantees Stability!

↳ $V(x,k+1) \leq V(x,k)$

Lyapunov

- Iterative LQR (for trajectory optimization)
  - Initial guess $x_0(t)$ $u_0(t)$

but using quadratic approx of nonlinear cost

$$g(x,u) \approx g(x_0,u_0) + \frac{\partial g}{\partial x}(x-x_0) + \frac{\partial g}{\partial u}(u-u_0)$$

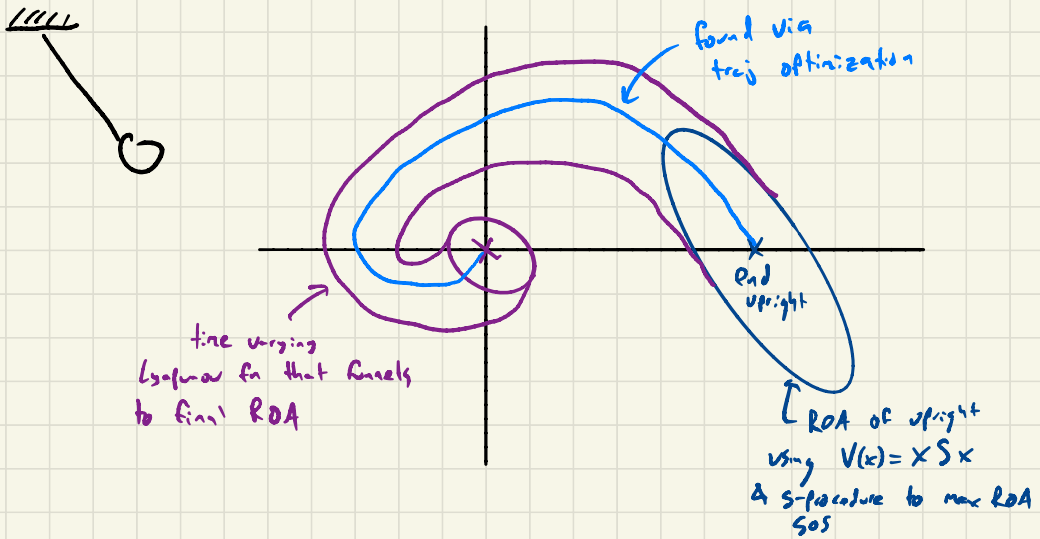
$$+ (x-x_0)^T \frac{\partial^2 g}{\partial x^2}(x-x_0) + \dots \quad \leftarrow \text{second order Taylor}$$

$$\min_u \int_0^T x^T Q x + q_1^T x + q_0 + u^T R u + r_1^T u + r_0 + u^T P x \, dt$$

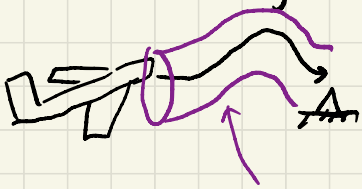
↳ Intuition: run offline to get trajectory
- Start w/ traj & iteratively run ILQR to find new traj to converge to nonlinear cost fn
- quadratic cost-to-go no longer @ origin
- iteratively find cost fn & relinearize to get new cost fn
- like SQP in SNOPT but more clever b/c using

↳ Intuition: MPC - if you can run optimization fast enough you can stabilize
ILQR - if you can stabilize non-trivial trajectory over & over, you can optimize

- Use case of TVLQR & SOS Lyapunov



- Use Case 2 Perching Plane



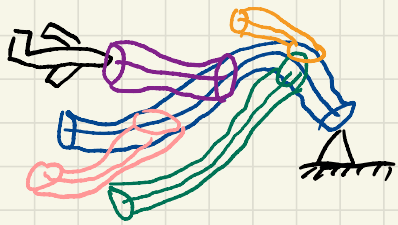
ROA via SOS

Downside: requires potentially small variation in init condition

Steps:

- System ID w/ motion capture to get nonlinear system dynamics
- Run offline through traj optimization
- Use TVLQR for online feedback stabilization
  - ↳ global control so can run fast (vs MPC)

↳ **LQR Trees**: Combats issue of requiring small initial set to stay in ROA funnel



↳ Fill space with funnels to converge to end ROA

↳ This allows linear fast feedback control (MPC not solution to everything)

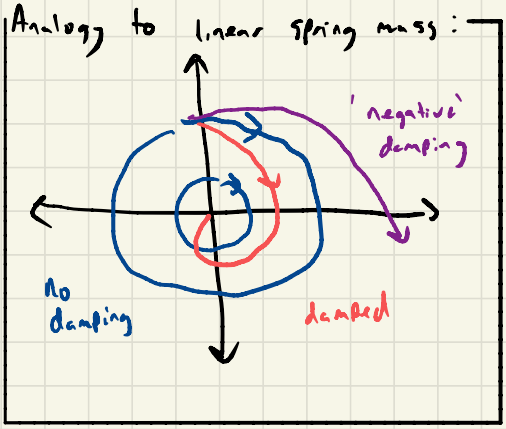
↳ Controlability analysis → loose control authority near end → adds flappers

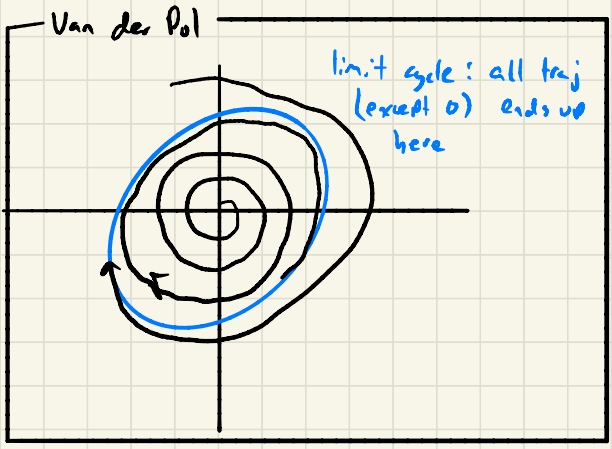
LECTURE 12: Simple Models of Walking

Key Ideas:
- Contact Mechanics - non-smooth mechanics
- Limit cycle stability

- Example (w/o impact) Van der Pol oscillator

$$\ddot{q} + \mu(q^2 - 1)\dot{q} + q = 0 \quad \mu > 0$$





- Stability of a Trajectory

$$\|x(t) - x^*(t)\| \to 0 \quad \forall\, x(t)$$

↳ False for Van der Pol b/c

P_1 & P_2 will never converge to same $x^*(t)$

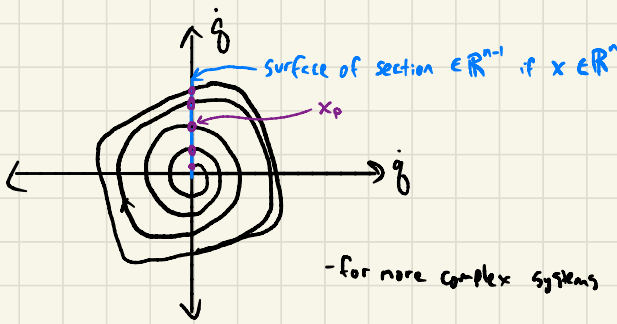
Limit Cycle Stability

- A periodic trajectory $\forall t = 0 \quad x^*(t) = x^*(t+T)$ is limit cycle stable iff (orbital stability)

$$\min_{\tau} \| x(t) - x^*(\tau) \| \to 0$$

↖ find closest point independent of time

- Surface of Section:



- for more complex systems will require numerical analysis

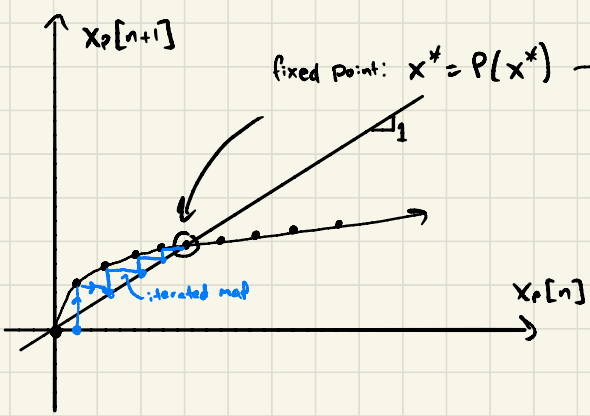
- Section is transverse to the flow
- all trajectories return to the section

$$x_p[n+1] = P(x_p[n])$$

$x_p \in \mathbb{R}^{n-1}$

Poincare Map: transforms limit cycle stability to classical analysis

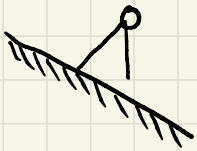
- Poincare Map for Van der Pol oscillator



fixed point: $x^* = P(x^*)$ → local stability via linearization

$$\frac{dP}{dx} \to x[n+1] = Ax[n]$$

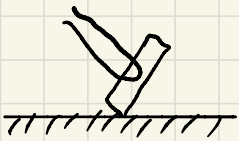
-Contact Models



Rigid vs **Soft**

Strict non-penetration

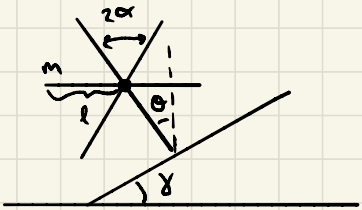
e.g. Spring damper model



Collisions modelled as impulse force

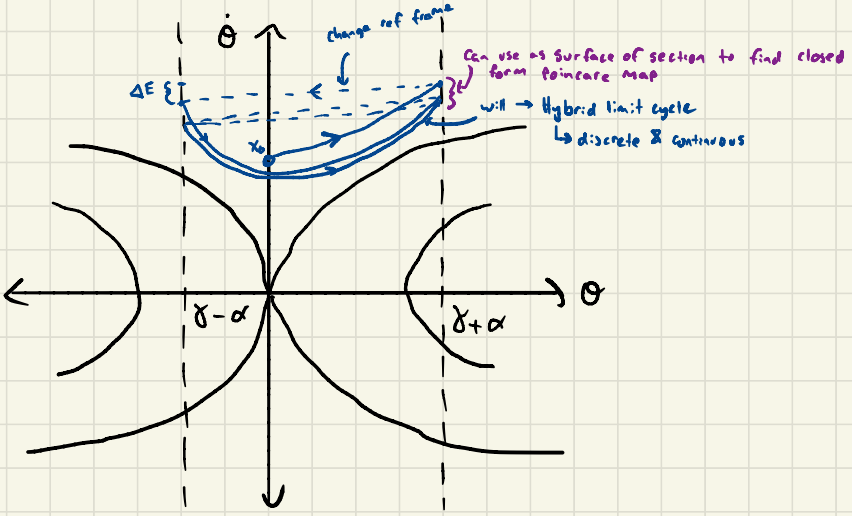
$\lambda_{normal} = -K(\text{penetration dist}) - \text{damp}$

-Rimless Wheel

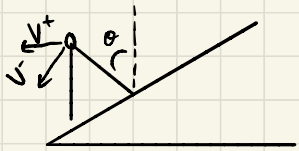


Assumptions:

- No slip (feet on ground as pin joint)
- Collisions are inelastic & impulsive (rigid)
  - ↳ no bouncing
- No double support



- Inelastic Collision

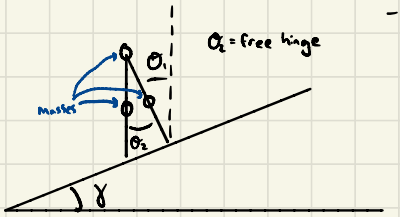


- All energy into ground is lost
- Angular momentum around point of collision is conserved

$$L^- = \ell \times m v^- = m\ell^2 \dot{\theta}^- \cos(2\alpha)$$
$$L^+ = \ell \times m v^+ = m\ell^2 \dot{\theta}^+$$
$$\therefore \dot{\theta}^+ = \dot{\theta}^- \cos(2\alpha)$$

- Side note: Ruina made this more efficient by having rear foot push off before impact → less $\Delta \vec{v}$ after collision (4x more efficent)

- Compass Gait



- Also has stable limit cycle but smaller

LECTURE 13 Simple Models of Running

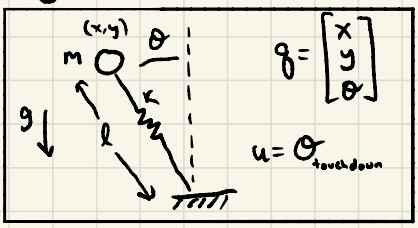
- Counterintuitively running (hopping) can be easier than walking

- Definition of Running:
  - Existence of an aerial phase
  - Exchange of energy eg.:

↳ Spring Loaded Inverted Pendulum (SLIP)

- Why Simple Models:
  - Tractable
  - Mechanical insights
  - Comparitive Biology
    ↳ Invariants in data → fundamental principles
  - As a template for higher DoF robots

- Spring Loaded Inverted Pendulum



Assumptions:
- Massless leg
  ↳ command θ instantaneously
- Perfectly elastic collision
  ↳ energy conserved
  (threat to stability)
- When foot on ground, pin joint
  (∞ sliding friction)

- Poincare Analysis
  - Apex-to-Apex map

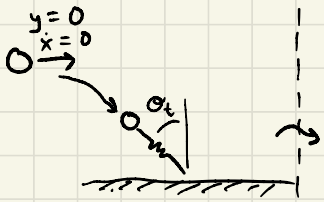
$$x = \begin{bmatrix} \cancel{x} \\ y \\ \cancel{\theta} \\ \dot{x} \\ \cancel{\dot{y}} \\ \cancel{\dot{\theta}} \end{bmatrix}$$

← Given const E $\dot{x} = f_n(y)$ $= 0$ at apex

$$y[n+1] = P_{u[n]}(y[n]) \qquad u[n] = \theta_{touchdown}$$

- Flight Phase

$$x = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \ddot{x} = \begin{bmatrix} 0 \\ -g \end{bmatrix}$$



$$y(t_{touchdown}) = l\cos\theta_{touchdown}$$

- Take-off to aerial

$x(0)$, $y(0)$, $\dot{x}(0)$, $\dot{y}(0)$ to $\dot{y}(t_{apex}) = 0$

+ Energy correction at apex (b/c linearized contact dynamics lose Cons E property)

- Standing Phase

$$x = \begin{bmatrix} r \\ \theta \\ \dot{r} \\ \dot{\theta} \end{bmatrix}$$

↳ "minimal coordinates" reduce constr. violation

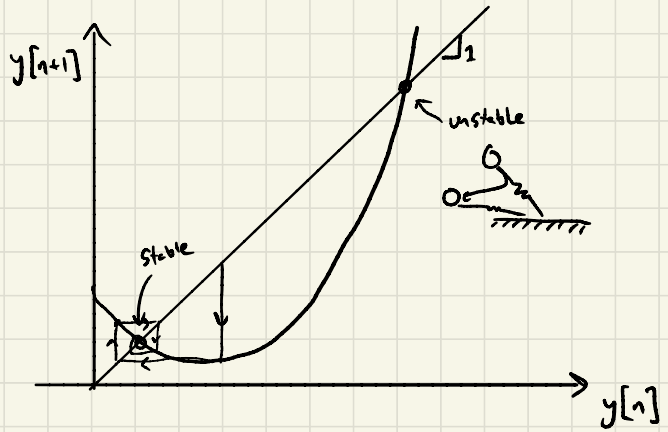
$$m\ddot{r} - mr\dot{\theta}^2 + mg\cos\theta - K(r_0 - r) = 0$$

$$mr^2\ddot{\theta} + 2mr\dot{r}\dot{\theta} - mgr\sin\theta = 0$$

↳ Hartmut Geyer gave us linearized about small angles

↳ No closed form solution

- SLIP approximate apex-to-apex map:



-SLIP Control

- choose $\theta_{touchdown}$ during aerial phase

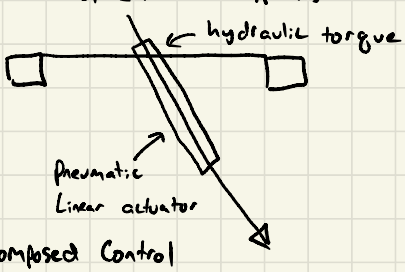
$y[n+1] = P(y[n], u[n])$

- Goal design $u[n] = \pi(y[n])$ to stabilize y^d
- Idea 1: find u^* s.t. $y^d = P(y^d, u^*)$
  Linearize P around (y^d, u^*) & do discrete time LQR
- Idea 2: Deadbeat Control (achieve desired y^d in 1 timestep) ↑ if it exists
  if P is invertible:

$$u[n] = P^{-1}(y^d, y[n])$$

↳ Can always set θ to this angle as fn y so it doesn't matter where the ground is. Cool!

-Continuous Control (Raibert Hoppers)



Decomposed Control

1) Hopping height (push at toe-off)
2) Foot touchdown to regulate speed
3) Stabilize attitude during stance

↳ early version of Boston Dynamic's Big Dog used this
↳ now a lot more optimization

LECTURE 14 Computational Methods For Legged Robots

- Simple models:
  - Rimless wheel (closed form soln Poincare map, ROA, limit cycles etc)
  - Compas gait (requires numerical tools to understand this & more complex)
  - SLIP (running) (requires small & approx to get Poincare map)
- New concepts
  - Limit cycle (stability)
  - Hybrid dynamics of contact
- Smooth System

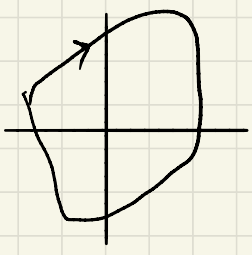
$\dot{x} = f(x)$ fixed pt is $f(x^*) = 0$

↳ may be hard to find but can use optimization

find x s.t. $f(x^*) = 0$

- Periodic soln → can also use trajectory optimization

- Ex Van der Pol oscillator (non hybrid case)



find x, t (knots in spline are decision var (cycle duration unknown)) s.t. $\dot{x}(t) = f(x)$ ← e.g. direct collocation

$x(0) = x(t_f)$

$x(0) = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix}$ ← avoid fp @ 0

- Ex Rimless wheel cycle w/ direct collocation (only 1 soln exists for this simple hybrid system)

$ml^2\ddot{\theta} - mgl\sin\theta = 0$ $\quad \dot{\theta}^+ = \cos(2\alpha)\dot{\theta}^-$ ← from impact

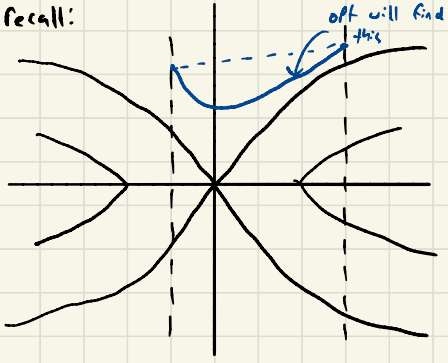
find x, t s.t. $\dot{x} = f(x)$ via dircol

$\dot{\theta}(0) = \gamma - \alpha$

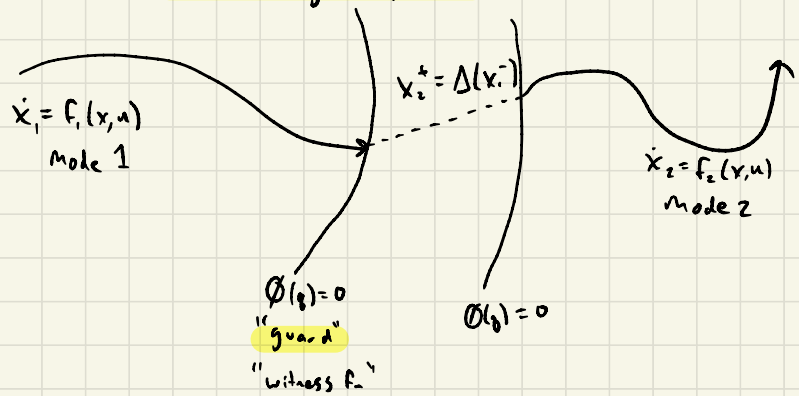
$\dot{\theta}(t_f) = \gamma + \alpha$

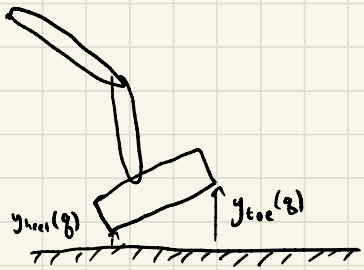
$\dot{\theta}(0) = \cos(2\alpha)\dot{\theta}(t_f)$

Recall:



- More General (autonomous hybrid systems)





- mode 1: foot in air
- mode 2: heel on ground
- mode 3: heel & toe on ground
- mode 4: toe on ground

- If you prescribe "mode schedule"

$x[0], \ldots, x[k]$ $\quad x[n+1] = f_1(x[n], u[n])$

$x[k+1], \ldots, x[N]$ $\quad x[n+1] = f_2(x_n, u_n)$

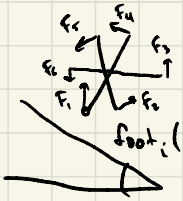
$\forall k \quad \phi(x[k]) > 0$

$\phi(x[K]) = 0$

$x[k+1] = \Delta(x[k])$

- This method breaks down for more complex systems when order of contacts are unknown

- "Minimal Coordinates" vs "maximal coordinates"



$q = \begin{bmatrix} x \\ z \\ \theta \end{bmatrix}$

$foot_i(q) \geq ramp$

Contact Jacobian

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = \tau_g(q) + Bu + \sum_i \underbrace{J_i(q)^T}_{\frac{\partial \phi(q)}{\partial q}} F_i$$

F_i: contact forces

if $\phi_i(q) > 0$, $F_i = 0$

if $\phi(q) = 0$

$\dot{\phi}(q) = 0$

$\ddot{\phi}(q) = 0 \Rightarrow$ Solve for F to impose the constraint

↳ {} time consuming to solve for every combination

Can solve w/ opt instead

find x, u, F s.t. $\dot{x} = f(x,u,F)$

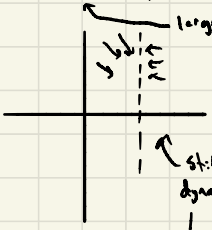
$\phi(q) = 0 \rightarrow \dot{\phi}(q) = 0$

- To avoid mode scheduling for complex systems
  - Idea 1: Soft Contact

F_i = soft-spring-model (x)

$= -K\phi(x) - \text{damping} \ldots$

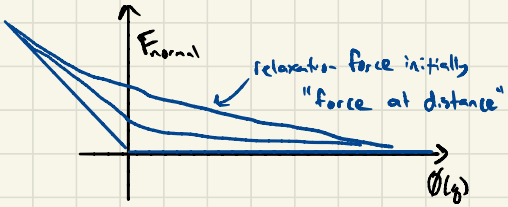
large to be accurate



stiff dynamics ⇒ finding best soln hard

↳ small Δt & many knot pts to solve

→ Emo Todorov uses this with relaxation to get soln easier



↳ issue w/ this is too much deformation could confuse collision engine

- Idea 2: Time Stepping approximation (Big Game engines w/ many objects)

$$x[n+1] = f(x[n], u[n], \lambda[n])$$

$\lambda[n]$ ← contact forces

↳ backwards Euler update

$x[n+1] = x[n] + dt\, f(x, u)$ ← forward euler

$\phi(q) \geq 0 \quad \lambda[n] \geq 0$

Complimentarity Constraint $\phi(q[n]) \cdot \lambda[n] \leq \epsilon = 0$

↳ same as $\phi(q) = 0$ or $\lambda = 0$

↳ Surprisingly Effective Solns via Linear Complementarity Problem LCP

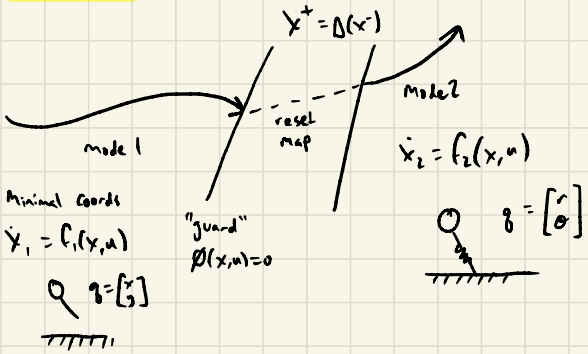
↳ "Let the Solve figure out which contact forces are active w/o soft contact"

↳ Russ uses this

LECTURE 15 Feedback & Lyapunov Through Contact

Last time:

Minimal Coord:



Maximal Coordinates: (hybrid case)

mode 1

$\dot{x} = f(x,u,\lambda)$

Contact Constraint forces

$g_1(x,u,\lambda) = 0$

$\lambda = 0$

mode 2

$\dot{x} = f(x,u,\lambda)$

$g_2(x,u,\lambda) = 0$

Hard Contact

$foot(q) = \text{Constant location} \quad \mathbb{R}^{2\times 1}$

$\frac{d\,foot}{dt}(q,\dot{q}) = 0$

$g_2(x,u,\lambda) \Rightarrow \frac{d^2\,foot}{dt^2}(q,\dot{q},u,\lambda) = 0$

if Soft Contact

$\lambda = -K\,foot(q) + \dots$

↳ For complex systems prefer maximal coords since less hand engineering coords

---

- **Trajectory Optimization w/ mode sequence** $\Rightarrow$ w/ Maximal Coords

  $\dot{x} = f(x,u,\lambda)$ $\quad \phi_i(q) \geq 0$ (e.g. height above ground) $\quad \lambda_i \geq 0$ (e.g. normal force) } hard Contact

  - Works w/ both min/max Coords
  - Very practical & works well

- **TO w/o mode sequence** $\Rightarrow$ **Soft Contact**

  $\lambda = \begin{cases} 0 & \phi(q) > 0 \\ -K\phi(q) & \phi(q) = 0 \end{cases}$ ↑ spring

  - "your mileage may vary"

  $\phi_i(q) \cdot \lambda_i = 0 \Rightarrow$ e.g. either touching ground ($\phi = 0$) or normal force $= 0$ ($\lambda = 0$)

- Sums of Squares for Lyapunov analysis through Contact

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = \tau_g(q) + Bu + \sum_i J(q)^T \lambda_i$$

(contact Jacobian; τ still polynomial)

$s = \sin\theta \quad c = \cos\theta$

$s^2 + c^2 = 1$

or

Taylor expansion to make linear

$\forall x, \lambda \quad V(q,\dot{q}) \geq 0 \quad \dot{V}(q,\dot{q}) \leq 0$

$V(q,\dot{q}) \leq \rho \quad \lambda \geq 0 \quad \phi(q) \geq 0 \Leftarrow$ S-procedure / Lagrange multipliers

$\uparrow$ R_{OA}

- SoS verification works well for simple polynomial controllers but not for e.g. MPC

- Stabilize a fixed point through Contact
  - Ex: Compass Gait balancing

Pin joint (minimal coord)

+LQR works ($\equiv$ Acrobot)

Maximal coords

$\dot{x} = f(x,u,\lambda)$

$g(x,u,\lambda) = 0 = foot(q)$

Can solve for λ & insert

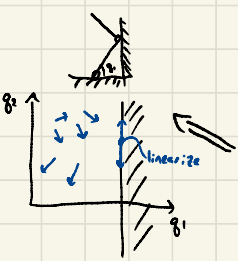
↳ $\dot{x} = f(x,u) \quad x \in \mathbb{R}^4$

$\approx Ax + Bu$

LQR(A,B,Q,R) will fail b/c not controllable

↳ intuit: linearizing around local point, loose contact behavior

↳ instead LP MPC will succeed from valid x (with $foot(q) = 0$)



Equality

- Constrained LQR

$\dot{x} = Ax + Bu$ Know additionally that $Gx = 0$

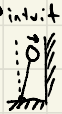
↳ project dynamics into null space of the (linearized) constraint (smaller state space) then project back up

- Prev Solns Stabilize for single state of contact but can't do e.g. 'push recovery' and constraint (contact) changes

↳ We are roughly stuck trying to solve across contacts

↳ Boston Dynamics does this with hand designed controllers

↳ intuit: LQR can't figure out bounce on wall to get back to fixed pt



↳ Soln: Mixed integer MPC (no global soln)

- Local stabilization across modes as mixed integer optimization

$$\min_{x,u} \sum x^T Q x + u^T R u \quad \text{s.t. } x[n+1] = Ax + Bu,\ x[0] = x_0,\ x[N] = x_{goal}$$

Now:

$x[n+1] = A^i x[n] + B^i u[n] + c^i$ when $D^i x + F^i u \leq 0$

↳ Piecewise affine

$$\min_{x,u,\lambda} \sum x^T Q x \ldots$$

$x[n+1] = Ax[n] + Bu[n] + B\lambda[n] + C \ldots$

$\lambda[n] \geq 0 \qquad Gx[n] \geq 0$

don't do $\lambda \cdot G = 0$ like before b/c not convex now

instead:

$\lambda_i[n] \leq b_i M$ ← Big positive constant "big M"

$G_i x[n] \leq (1 - b_i) M$

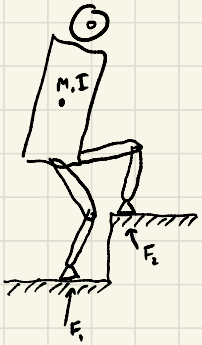
$b_i \in \{0, 1\}$

↳ Mixed integer optimization

- Note Mixed integer always non-convex
- Worst case complexity is very bad
- "Mixed-integer convex" iff ignoring integer results in convex
  - ↳ get lower bounds ⇒ effective branch & bound search
  - ↳ very efficient solvers:
    - return global optimal or infeasible
- Usecase example: Atlas robot given set of foot placements determines which foot to place in which placement at which time
  - interesting: Atlas pushed against wall & using push recovery is unreliable with full non-linear system but consistent w/ MI convex but still only works near upright (b/c linearized)
- Not realistic to run in real time
  - Some methods for faster compute:
    - tighter formulation (from disjunctive programming)
    - Approx explicit MPC: assume contact x will be activated & find ROA set of initial conditions for recovering using that
    - Lyapunov based synthesis

LECTURE 16 Humanoids "Highly-articulated legged robots"

- So far: traj opt, LQR, SOS still useful w/ legs
- Today: as complexity increases so does computation, but at some point it becomes easy again

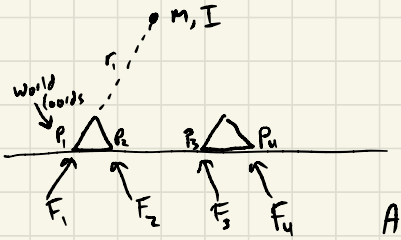


$F_{i,z} \geq 0 \quad |F_{i,x}| \leq \mu F_{i,z}$

- doesn't need to be massless legs
  - ↳ Can regulate COM position & total angular momentum of COM
- Ignoring impact is possible:
  $z_{com} = const \Rightarrow \ddot{z} = 0$ & $\ddot{\theta} = 0$
  - ↳ implicitly impact no energy (foot speed at impact $= 0$)

- Flat Terrain

LMB: $m\ddot{x} = \sum F_{i,x} \qquad m\ddot{z} = \sum F_{i,z}$



$F_{i,z} \geq 0 \Rightarrow \ddot{z} \geq -g$
$|F_{i,x}| \leq \mu F_{i,z} \Rightarrow |\ddot{x}| \leq \mu(\ddot{z} + g)$
} limits possible trajectories of COM

AMB: $I\ddot{\theta} = \sum (r_{i,z} F_{i,x} - r_{i,x} F_{i,z})$

$r_i = p_i - \begin{bmatrix} x \\ z \end{bmatrix}$

$x_{com} = \frac{\sum x_i m_i}{\sum m_i} \in \mathbb{R}^3$

$x_{cop} = \frac{\sum p_{i,x} F_{i,z}}{\sum F_{i,z}} \Rightarrow x_{cop} \in$ Convex hull (p) "support polygon"
(center of pressure; replace all forces with one)

↳ $(m\ddot{z} + mg)(x_{cop} - x) = (z_{cop} - z) m\ddot{x} - I\ddot{\theta}$

↳ Assume $\ddot{\theta} = 0$, $\ddot{z} = 0$, $\dot{z} = 0$ $\Rightarrow z - z_{cop} =$ height

↳ $\ddot{x} = \frac{g}{h}(x - x_{cop})$ ⇐ AKA COM motion dictates what COP needs to be

↳ important! in argument of force vs. position control
this says you can't control interaction forces even w/ position control (how Asimo works)

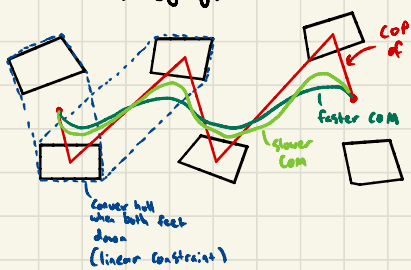
↳ works for 3D too

- **Zero Moment Point (ZMP)** : think center of pressure

- In flat ground walking x_{CoP} is a Zero-Moment Point
- requires 'light' footsteps (zero speed of foot at contact for ZMP to hold)

- **Three Stage Walking Plan/Control**
  1) Footstep Planning : MI Convex Optimization
  2) Plan COM/COP : LP/QP
  3) Fill in details $(q, \dot{q})$

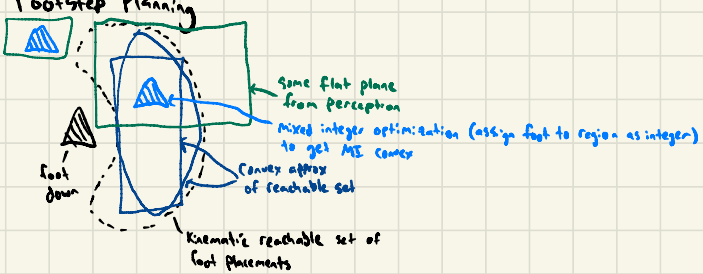
} Simple models, only care about COM/COP/footsteps



- COM doesn't need to be w/in support polygon to walk w/ ZMP
  - ↳ will fall if stop midstep
  - ↳ slow walking means COM always w/in support polygon

- Can use linear optimal control to jointly solve COP/COM trajectories since linear eqns & linear constraints
  - Eg. give desired traj but then minimize effort via quadratic cost
  - Can even use LQR to do this

- **Footstep Planning**



- MI-Convex opt based
- Nonlinear Traj opt based
- Sampling based

} all use simplified kinematics & almost no dynamics

- Hard to do footstep planning while optimizing COM/COP because it becomes bilinear (no longer convex MI) : $\tau = p \times F = p_z f_x - p_x f_z$ ⇐ bilinear
  - ↳ current research looking into this

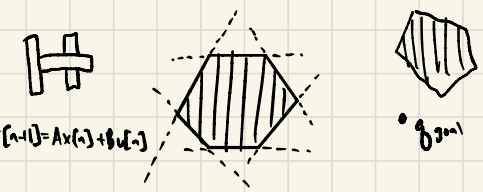
LECTURE 17: "COMPLETE" PLANNING

- Trajectory optimization not always best for planning if NLP
  - Not robust
  - No guaranteed soln
  - May not be optimal

- "Complete" (AI/Graph Search Community)
  - Def: guaranteed to find a soln if one exists
  - "Global optimal" if it finds the optimal plan

1) Decompose non-convexity in problem & search all decompositions
2) Randomized Motion Planning: very good but not great for dynamical systems

} will find something in between



$$\min \sum (x[n]-x_{goal})^T Q (x[n]-x_{goal})$$

Interior polygon described by linear constraints

$$P \{x \mid Ax \le b\}$$

Exterior: $a_1 x \ge b$ OR $a_2 x \ge b$ OR $a_3 x \ge b$

"breaks everything" requires Mixed Integer

↳ Disjunctive constraints w/ integer variables (binary)

$$c_i[n] \in \{0,1\}$$

$$a_i x[n] \ge b - c_i[n] M$$

↑ large constant

$$\sum c_i[n] \ge 1$$ ← MI at least one free space occupied

↳ MI opt uses branch and bound which uses relaxed constraints e.g. $c_i[n] \in [0,1]$

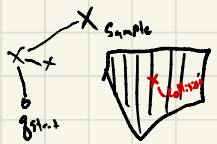
↳ best when using MI convex

↖ Can warmstart w/ feasible guess
Solns help prune branch & bound

↳ MI can solve any NP-hard problem so takes a while

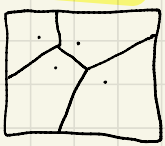
↳ Make easier (non-MI) by breaking up free space into convex polygons (decomposition)
- not complete if convex free space can't capture everything

- Randomized Motion Planning "Sample based Planning"
  - RRT: works well for high dimensions, explores rapidly



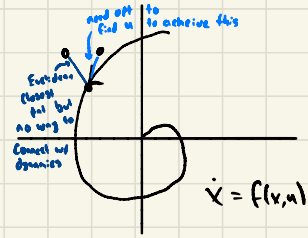
1) Random sample
2) find closest point on current tree
3) grow tree towards sample point

- has a Voronoi bias



← highest probability of exploring largest/least dense Voronoi regions

- Best for kinematic problems (any direction is feasible) (ignore dynamics)
  - As $t \to \infty$: uniform coverage of free space
  - Even pendulum dynamic constraints make it hard to find soln

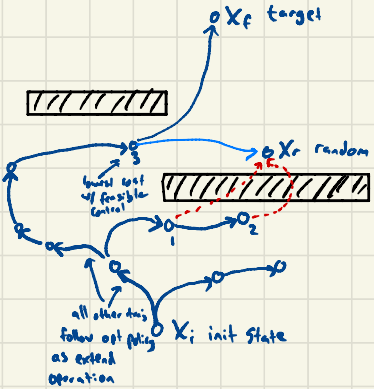


1) Better Sampling distributions
2) Better distance metric
3) Better extend (e.g. Traj optimization)

Approx
"Distance" Metrics:
- Min time to go (bounded value Prob)
- Cost-to-Go
  - Eg. approx by linearizing around sample & find LQR cost-to-go (LQR RRT*)

Cool! Sample based planning for dynamical systems
"To this day one of the most satisfying solns"

- Emilio Frazzoli method! 2002
  - For real-time motion planning of helicopter
  - Value iteration on mesh/ grid of obstacle-free space to get cost-to-go $J^*(\cdot, X_f)$
    - ↳ use as approx of distance metric in RRT for real-time RRT
    - ↳ use pre-computed optimal policy $\pi(\cdot, X_f)$ when no obstacle



- ↳ exploit invariants in configs to get value iteration of 6 DoF helicopter

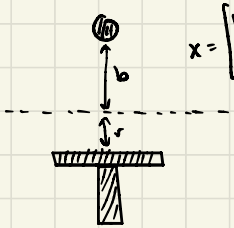
- Kinodynamic Planners
  - Control based Planners OMPL
    - For better Sampling (address #1)
      - Reachability Guided RRT: grow 2 trees (normal & reachability)
- Conclusion: Russ thinks ideal planner will lie somewhere in between MI opt & RRT methods

LECTURE 18: FEEDBACK MOTION PLANNING

- Last Time: "Complete": given start & goal, will find path if it exists
- Getting to Feedback:
  - Plan every timestamp (MPC)
  - Roadmaps: graph across map can be reused with various start & goal
  - RRT* (backwards Trees)
    - gets rid of kinks/RRT dance by constantly refining tree
    - works well but can't handle disturbances

Sidenote: "RRT dance" funny set of intermediate configs before getting to final

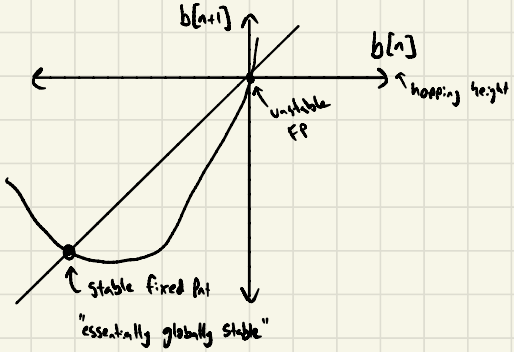
- Simplest Juggling Model "Line Juggler"



$x = \begin{bmatrix} b \\ \dot{b} \end{bmatrix}$ $u = [\dot{r}]$

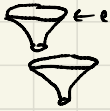
Assumptions
- instantaneous elastic collisions (w/ coeff of restitution)

Mirror Law:
- impose $\dot{r}(t) = -k\dot{b}(t)$
  - ↳ collisions occur at $r=0$ $b=0$ $\dot{b}<0$
  - ↳ one-dimensional Poincaré map



⇓

- "Sequential Composition of Dynamically Dexterous Robot Behaviors" - R.R. Burridge
  - initial idea of combining Lyapunov fns from simple controllers to have ROA fill state space



- before SoS so used experimental data to find ROAs

- Probabilistic Feedback Coverage
- Lyapunov fns along trajectories

LQR Trees

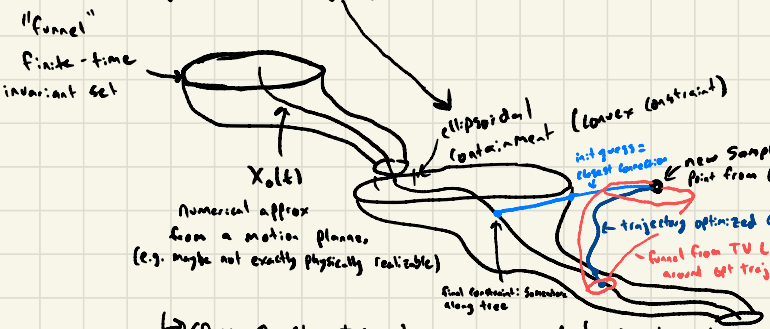
$\dot{\bar{x}}(t) = A(t)\bar{x}(t) + B\bar{u}(t) \Rightarrow \bar{u}(t) = -K\bar{x}(t)$ ← TV LQR

$J(x,t) = \bar{x}^T S(t) \bar{x}$ ← use TV cost-to-go as Lyapunov fn

Trajectory $x_0(t)$ defined over some finite $t = [0, t_f]$

Find $\rho(t)$ ← cubic spline s.t. $\dot{J}(x,t) < \dot{\rho}(t)$ when $J(x,t) = \rho(t)$ ← Note! normally say $J \leq \rho(t)$, instead only look at wall of funnel (within funnel)

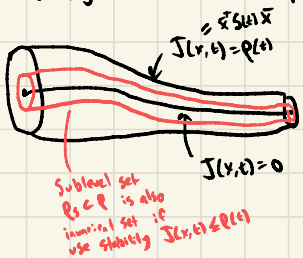
$J(x, t_f) \leq \rho(t_f)$ to be inside next Controller



↳ Sparse sampling tree because no need to sample w/in existing funnels

↳ Think: different Controllers that turn on for diff parts of state space (depending which funnel is active) like a state-machine

- Issue: building RRT LQR tree is expensive → offline Computation. How run online with obstacles?



$\dot{J}(x,t) \leq \dot{\rho}(t)$ → $\forall x$ $J(x,t) \leq \rho(t)$ ← Stability (hard to do)

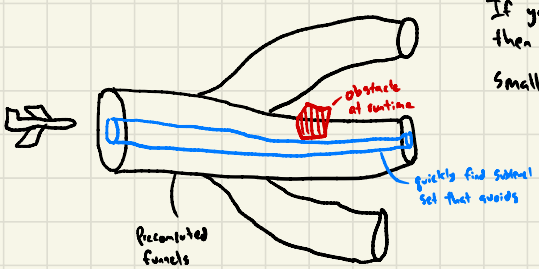
$\forall x$ $J(x,t) = \rho(t)$ ← finite time invariant

$\forall x$ $\alpha\rho(t) \leq J(x,t) \leq \rho(t)$

$0 \leq \alpha \leq 1$ ← avoid 0 & numerical tolerances

If you use these stability measures instead, then can quickly shrink $\rho(t)$ at runtime to find smaller funnel sols. This is useful for online

Ex:



- What if you want larger ROA than given by TV LQR applied to non-linear system:

Can find new control that optimizes ρ

$\dot{x} = f_1(x) + f_2(x)u \quad u = -Kx$ or $u = -K m(x)$ ← monomials

Given $V \quad \dot{V}(x) = \frac{\partial V}{\partial x}\left[f_1(x) + f_2(x) K m(x)\right]$

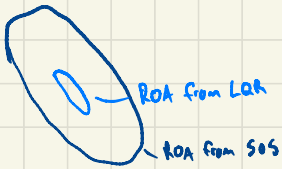
$\dot{V}(x) \leq 0$ for all x $V(x) \leq \rho$

↳ use Sums-of-Squares Programming to search for ρ & K simultaneously

↳ Can start w/ K from LQR

- find V to get largest ρ
- lock in V & find K to get even bigger ρ
- and alternate (Bilinear Alternations to maximize ρ)

↳ note: can't search for V & K that maximize ρ at same time



↳ LQR Trees great for vehicles/acrobot/cartpoles/etc but NOT manipulation

because full state feedback which is not possible w/ manipulation

- shouldn't need to know full state of key-ring to pick them up
- this is reason Russ has become interested in manipulation

LECTURE 19 MODEL SYSTEMS WITH STOCHASTICITY (Robust Control)

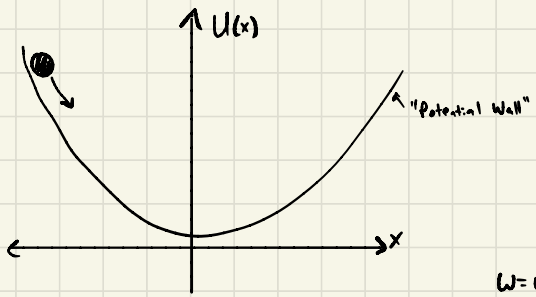
So far $\dot{x} = f(x,u)$ $\quad x[n+1] = f(x[n], u[n])$

$\dot{x} = f(x,u,w)$ $\quad x[n+1] = f(x[n], u[n], w[n])$

$w(t)$ or $w[n]$ output of some random process

- $w[n]$
  - disturbances
  - model uncertainty (process noise)
  - Not measurement noise

- Additive Noise $\quad x[n+1] = f(x[n], u[n]) + B_w w[n]$
- Ex Particle in Bowl with Brownian Motion



$$U(x) = \frac{1}{2}\alpha x^2$$

$$x[n+1] - x[n] = -\frac{\partial U}{\partial x}\Big|_{x[n]} + w[n]$$

$$x[n+1] = (1-\alpha)x[n] + w[n]$$

$w = 0$ fixed point $x^* = 0$

expo stable for $0 < \alpha < 2$

- $w[n]$ is Gaussian white noise

$$E[w[n]] = 0$$

$$E[w[i]w[j]] = \begin{cases} \sigma^2 & j = i \\ 0 & \text{otherwise} \end{cases}$$

- Given $x[0]$ what is $x[n]$ what is $x[\infty]$?

$x[n]$ is a random variable described by

$P_{x[n]}(\cdot)$ density function

$$P_w(w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-w^2}{2\sigma^2}} \leftarrow \text{Gaussian}$$

$$\hookrightarrow P_{x[n+1]|x[n]}(x[n+1] \mid x[n]) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{-(y-(1-\alpha)x)^2}{2\sigma^2}$$

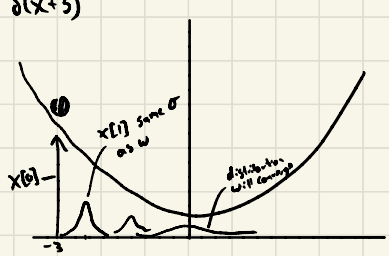
"Master Eqn" $\quad P_{x[n+1]}(y) = \int_{-\infty}^{\infty} P_{x[n+1]|x[n]}(y|x)\, P_{x[n]}(x)\, dx$

↑ output remains Gaussian; ↑ convolution

- Note init condition $x[0] = 3$

treat probability like delta fn

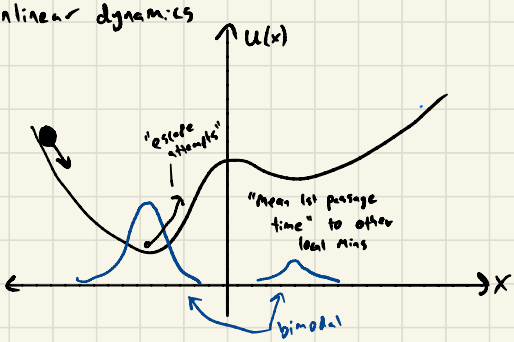
$$P_{x[0]}(x) = \delta(x+3)$$

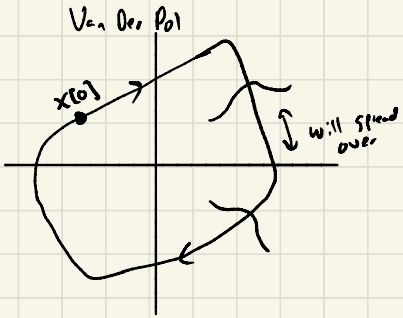


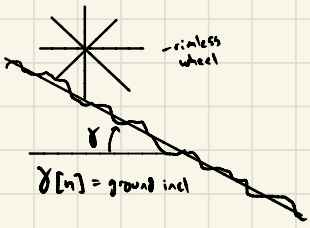
- Steady State distribution $P_{x^*}(\cdot)$

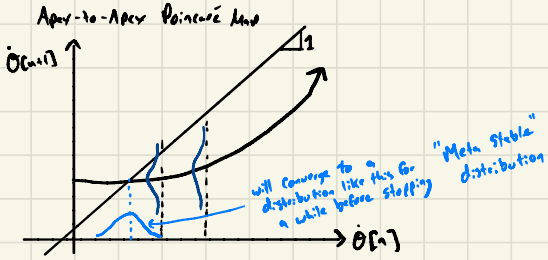
$$\mu^* = 0 \quad (\sigma^*)^2 = \frac{\sigma^2}{2\alpha - \alpha^2}$$

- Nonlinear dynamics









- SideNote: When to worry about Stochasticity?
  - ↳ when noise distro is relatively wide compared steepness of dynamics

Stationary distribution is standing still

Since inescapable once there

↳ Think: "Any system w/ Stochasticity will fall over given enough time (for long tail noise)"

↳ how to control system if final state is always fall down?

↳ "Meta-Stable" distribution - long living transients in $P_{x[n]}(\cdot)$

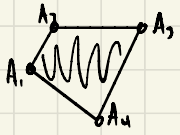
1) Worst case design/analysis

Typically for bounded uncertainty

$P_w(\cdot)$ has finite domain (unlike Gaussian: tails → ∞)

e.g. via Common Lyapunov Function

$\dot{x} = Ax \quad A = \sum_i \beta_i A_i$ affine combination

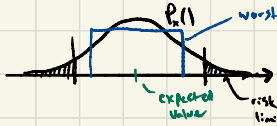


for all w, $\dot{V} < 0$

↳ often too conservative since considering worst case

2) Stochastic Control

- Write objective on $P_{x[n]}(\cdot)$ most common is expected performance
  - Another example is **Risk Aware Control** (chance constraints)



↳ Sometimes can use convex opt other times gets hard fast

  - Expected Value (used in Reinforcement Learning)

$$J = \sum_{n=0}^{\infty} g(x,u) \quad \text{now min} \quad E[J] = \sum_{n=0}^{\infty} E[g(x,u)]$$

↳ now J is random variable

$$= \sum_{n=0}^{\infty} \left[\int_x g(x,u) P_x(\cdot) dx \right]$$

- **Stochastic LQR**

$$x[n+1] = Ax[n] + Bu[n] + B_w w[n]$$

↑ non-zero Gaussian iid

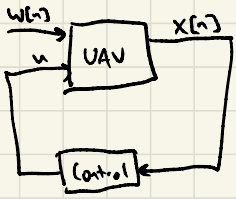
$$\text{Min } E\left[\sum^{\infty} x^T Q x + u^T R u\right]$$

$u = -kx$ which is the **same K in normal LQR!** ← not same for nonlinear

$J = \infty = x^T S x + C$

↑ original S ⇐ goes to ∞ $\sum E[w[n] w^T[n]]$

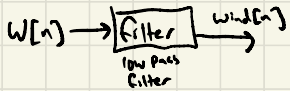
Ex UAV in wind

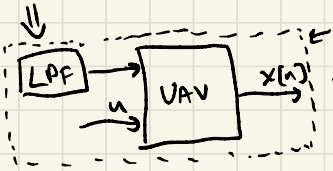


Sidenote: LQR doesn't provide guarantees for robustness

(Funny paper says there are no guaranteed stability margins for LQG)

Wind is not Gaussian iid





← do stochastic LQR on this system "whitening filter"

$u = -Kx$

↑ UAV + wind estimation now gives good robustness

LECTURE 20 Robust Constrained Control

$x_{t+1} = f_t(x_t, u_t) + w_t$ (with constraints on u, x)

w_t: disturbance

$w_t \in W_t$

$u_t = \Pi_t(\overbrace{x_0 \dots x_t, u_0 \dots u_{t-1}}^{\text{Controller w/ memory}})$

Π_t: control policy

- Problem: design Control Policy Π such that

all possible trajectories is subset of acceptable trajectories

$\mathcal{O}(x; \Pi)$ ← trajectory with each run bc w_t $\subseteq$ Ψ

- From now on focus on linear systems

$x_{t+1} = A_t x_t + B_t u_t + w_t \quad w_t \in W_t$

Polytope aside:

- H-Polyhedron: define edges

$Hx \leq h$

- good in high dimensions ($2n$) faces
- easy to check if within (check inequality)

Polytope = bounded

Polyhedron = unbounded

- V-Polytope: convex hull of vertices $\{v_1 \dots v_N\}$

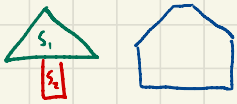
$x = \sum_{i=1}^{N} \lambda_i v_i \quad \lambda_i \geq 0 \quad \sum_{i=1}^{N} \lambda_i = 1$ ← shape

- bad in high dimensions 2^n vertices
- hard to check if in within (solve linear program)
- easy to get Minkowski sum

Minkowski Sum

$S_1 \; S_2 \; C \; R^n$ ← in

$S_1 \oplus S_2 = s_1 + s_2 \quad s_1 \in S_1 \quad s_2 \in S_2$



- $x_{t+1} = A_t x_t + B_t u_t + \underbrace{w_t}_{c_t \text{ const}}$ $\quad w_t \in \underbrace{W_t}_{\text{set}}$ ← non-deterministic

$\mathbb{P}_t = \bar{x}_t + G_t Q$ (G_t: generator)

$\bar{x}_t$: center; Q: typically "cube": $[-1, 1]$

Zonotope $Z(\bar{x}, G)$

↑ symmetric about center e.g.

- Controller:

$x = \bar{x}_t + G_t q \qquad q \in Q$

$u_t = \bar{u}_t + \Theta_t q$

↑ design parameter feedback gain

now $A_t x_t + B_t u_t \Rightarrow A_t(\bar{x}_t + G_t q) + B_t(\bar{u}_t + \Theta_t q)$

$= A_t \bar{x}_t + B\bar{u}_t + C_t + (A_t G_t + B_t \Theta_t) q$

$\mathbb{P}_{t+1} = \underbrace{(A_t x_t + B_t u_t + C_t)}_{\bar{x}_{t+1}} + \underbrace{(A_t G_t + B_t \Theta_t)}_{G_{t+1}} Q$

Linear

- Can solve w/ Trajectory Optimization (Linear Program)

$\underbrace{\bar{x}_t \; \bar{u}_t \; G_t \; \Theta_t}_{\text{decision vars } t=0 \ldots T-1}$ given $\mathbb{P}_T \subseteq X_{goal}$ ↖ linear constraints not shown here

Dynamics

x, u constraints: $\mathbb{P}_t \; x \mathbb{H}_t \subseteq \mathbb{H}_t$

↑ interesting note:

designing trajectory (x, u) & feedback-gain policy simultaneously (G, Θ)

- Redo ↑ but for non-deterministic

$$x_{t+1} = A_t x_t + B_t u_t + w_t \quad w_t \in W_t = \underset{\uparrow \text{Zonotope}}{Z}(\bar{w}_t, \underline{W}_t)$$

$$P_t = Z(\bar{x}_t, G_t)$$
$$x_t = \bar{x}_t + G_t \xi$$
$$u_t = \bar{u}_t + \Theta_t \xi$$

$$x_{t+1} = (A_t \bar{x}_t + B_t \bar{u}_t + \bar{w}_t) + \underbrace{(A_t G_t + B_t \Theta_t)\xi + w_\xi'}$$

Minkowski Sum for Zonotope

$$Z(\bar{x}_1, G_1) \oplus Z(\bar{x}_2, G_2) = Z(\bar{x}_1 + \bar{x}_2, (G_1 | G_2)$$

- We will expand Polytopes, need to find Θ to shrink down
- Same opt problem as before but Generator is longer

Robust Control

- Invariance: start w/ in set & need to stay within for $t \to \infty$ w/ disturbances

e.g. 

$\theta \in [\theta_{min}, \theta_{max}]$

- Can Create LQR Tree equivalent (for linear systems & handles disturbances & constraints) by filling up state space with Polytopes

LECTURE 21 Manipulation (MIT 6.881: Russ new class on manipulation)

- Guest Lecture

Dynamic Programming
- Low DoF
- $f(), x, u, C()$ known

Stability analysis with SOS
- $f(), x, u$ is known
- $f()$ is vector-valued polynomial
- used for stability analysis of commercial planes

Trajectory Opt
- $C(), f(), x, u$ known
- solutions only locally opt
- used for SpaceX rocket landing

→ falls apart for manipulation (e.g. what is state for making Sushi)
- unknown $f(), C()$ (objective), $g()$ (observation fn), x

- Imitation Learning: learn policy π from expert by observing actions & trajectories

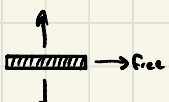
LECTURE 22: Reinforcement Learning

- **Reinforcement Learning**: collection of algorithms for "black box optimization" of stochastic optimal control problems
  - Berkeley: "ad-hoc but strangely successful" ← lol

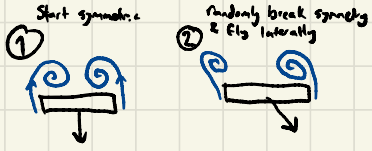
black box optimization:
Only have access to $\sum g(x,u)$ costs
don't have $f(x,u)$ (model free)
how to optimize?
↳ Russ is skeptical. Should use model if have access
↳ e.g. Cooper manipulation hard to model

Stochastic optimal control:
$J(x,u) \Rightarrow E[\sum_{n=0}^{N} g(x,u)]$
what we optimize
↳ Russ is excited about this part
↳ more generalizable

- **Example** Fluid dynamics of flapping
  The heaving foil (Jun Zhang NYU)



Symmetric flat plate



↳ used RL to optimize flight efficiency w/out model

← dynamics governed by PDE (Navier Stokes)
↳ can't write $f(x,u)$
CFD very expensive
↳ good case for RL

- **The Policy Gradient Trick** (REINFORCE)

$\min_{\alpha} E[g(x)]$ with $x \sim p_\alpha(x)$ ← policy

x is random variable that depends on parameters α
↑ trajectory

$$\frac{\partial}{\partial \alpha} E[g(x)] = \frac{\partial}{\partial \alpha} \int dx\, g(x) p_\alpha(x)$$
$$= \int dx\, g(x) \frac{\partial}{\partial \alpha} p_\alpha(x)$$
$$= \int dx\, g(x) p_\alpha(x) \frac{1}{\partial \alpha} \log p_\alpha(x)$$
$$= E\left[g(x) \frac{1}{\partial \alpha} \log p_\alpha(x) \right]$$

aside: $y = \log u$
$\frac{\partial y}{\partial x} = \frac{1}{u} \frac{\partial u}{\partial x}$
$\frac{\partial}{\partial \alpha} \log p_\alpha(x) = \frac{1}{p_\alpha(x)} \frac{\partial}{\partial \alpha} p_\alpha(x)$

approx w/ Monte-Carlo
$$\frac{\partial}{\partial \alpha} E[\sum_{n=0}^{N} g(x_n)] \approx \frac{1}{N} \sum_{n=0}^{N} g(x_n) \frac{\partial}{\partial \alpha} \log p_\alpha(x_n)$$

↑ gives direction to update parameters α after running many trials

Caviat: true of expected value is a weak statement

↳ Think: Surprising! gradient of performance depends on cost & gradient of controller but not on dynamics / init conditions

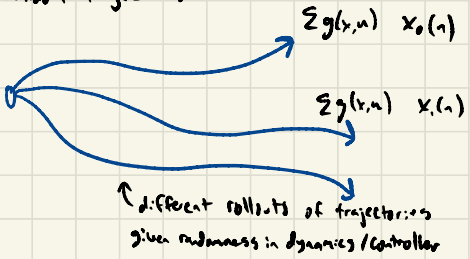
Rewritten in slides as:

$$\frac{\partial}{\partial \alpha} E\left[\sum_{n=0}^{N} g(x[n], u[n])\right] = E\left[\sum_{n=0}^{N} g(x[n], u[n]) \sum_{n=0}^{n} \frac{\partial}{\partial \alpha} \log P_\alpha(u[k] | x[k])\right]$$

↑ e.g. cost on stage #3 only influenced by stages 0:3

- **Intuition:**

N random trajectories



e.g. Simple Controller $u = -Kx + w$

↑ random number in policy helps explore space

↳ update based on policy gradient
update K so that rollouts with w giving low costs
are more likely in the future

↳ **inefficient update**: Many Monte-Carlo rollouts to get good policy

↳ Same update can be used for simple spline parameterization of trajectory or deep neural network

Black-box Optimization:

$$\min_{\alpha} g(\alpha)$$

How to do gradient-free optimization?

- Idea: **finite differences**

$$\frac{\partial g}{\partial \alpha}\Big|_{\alpha} \approx \left[\frac{g(\alpha + \epsilon_i) - g(\alpha)}{\epsilon}\right] \quad \epsilon_i = \begin{bmatrix} 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix}$$

↑ n parameters

↑ n+1 evaluations ← Can be expensive

- Then do gradient descent: $\alpha[k+1] = \alpha[k] - \eta \frac{\partial g}{\partial \alpha}\Big|_{\alpha[n]}$

- **Stochastic gradient descent**:

- as long as going "downhill" can get away w/ less eval of $g(\alpha)$

- Simpler idea: **"Weight Perturbation"**

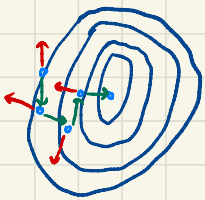
$$\Delta\alpha = -\eta\left[g(\alpha + \beta) - g(\alpha)\right]\beta$$

↑ small random vector

↑ 2 evaluations

↑ performs stochastic gradient descent

↑ choose 1 of 2 directions



$$E[\Delta\alpha] = -\eta\sigma^2 \frac{\partial g^T}{\partial \alpha}$$

↳ expected avg is in direction of actual gradient

- Often wins over finite difference over # eval to reach local min
- Similar idea: "Trust Region Optimization" bound step size

- Even simpler estimate

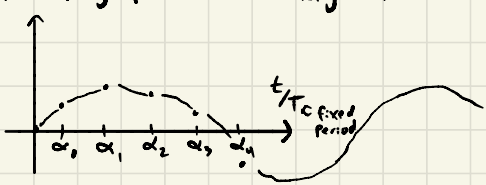
$$\Delta\alpha = \frac{-\eta}{\sigma^2}[g(\alpha+\beta) - b]\beta$$

↳ baseline "expected performance"

- Still stochastic gradient descent: $E[\Delta\alpha] \propto -\frac{\partial g^T}{\partial \alpha}$
- Cool: simple algo w/ strong property
- Uncool: this is a weak property (same property as policy gradient trick)

Key limitation of RL today: "Sample Complexity"

Example Policy params for hearing foil



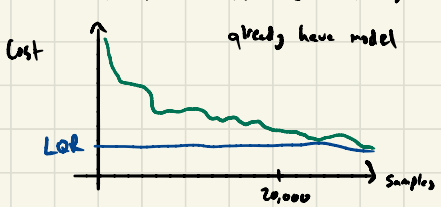
↳ better than parameterizing by Fourier coefs: better to have all params contribute about the same to cost

- Cost: Max inverse cost of transport
  - converge consistently to triangle wave 

"extremum seeking control" "iterative learning control"

- works well for repetitive experiments

- Example when not to use RL: Quadrotor stabilization



- Given accurate sim & 1000s of sim yrs can do robot hand manipulation
- Takeaway: current state takes too many trials. Only marginally better than running billions of random trials and taking best one
  - still use model when available

Lecture 23: Reinforcement Learning II

- RL is black box optimization for stochastic optimal control
- Last time: Policy Gradient methods

$u = \pi_\theta(x)$ (θ: parameters; search for opt parameters)

$\dot{x} = f(x,u)$

$y = h(x,u)$

e.g.
- $u = -Kx$ (search for K)
- $u = DNN(x)$
- $u = -\pi_\theta(y)$ (output) "output feedback"
  - ↳ powerful, unlike e.g. LQR: full-state feedback

↳ Gradient descent: trial & error → slow

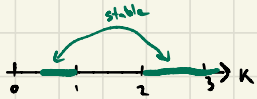
- Two Main Concerns
  1) Sample complexity: e.g. 20,000 iterations for quadrotor
  2) Non-Convexity

$\dot{x} = Ax + Bu$

$y = Cx$

$u = -Ky$

$A = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad C = [1\ 1\ 3]$



↳ disconnected set of Controllers

↳ Intuit: Policy gradient search for K may fail to jump to find global min

↳ deep NN w/ many params can overcome local min by always finding new direction w/ lower cost

- Today - Q-Learning
  - Actor-Critic

Idea: Learning Value Functions

Policy Gradient ← inefficient — Dynamic Programming → efficient

↳ eval dynamics exactly once from terminal conditions back to $t=0$
↳ requires model

Options

Today ↓

① Learn $f(x,u)$ — System ID + Model based Control

② Learn $\pi(x)$ — Policy Search

③ Learn $\hat{J}(x)$ — Cost-to-go

$J^*(x)$ ↳ optimal

$J^\pi(x)$ ↳ for some policy

$\hat{J}(x)$ ↳ approx

$$J(x) = E\left[\sum_{n=0}^{\infty} \gamma^n g(x[n], u[n])\right]$$

∞: horizon

γ: ↳ discount; $0 < \gamma < 1$; keep bounded & value newer in time more

$$x[n+1] = f(x[n], u[n], w[n])$$

- Learn only $\hat{J}^*(x)$:

$$J^*(x) = \min_u \left[g(x,u) + \gamma E[J^*(f(x,u))] \right]$$

$$\pi^*(x) = \arg\min_u [\dots]$$

↳ given cost-to-go also have policy (if know $f(x,u)$)

not available in RL model free

- Two Work Arounds (not having $f(x,u)$ to map $J^* \to \pi^*$)
  1) Q-Learning $Q^*(x,u)$ instead of $J(x)$
  2) Actor-Critic Learn $\pi(x)$ & $J^\pi(x)$ simultaneously

- Q-function
  - Define $Q^\pi(x,u) = E[g(x,u) + J^\pi(f(x,u))]$ ⇒ think: cost of single step control + remaining cost-to-go

    $J^*(x) = \min_u Q^*(x,u)$

    J has $\dim(x)$ inputs (scalar output)

    Q has $\dim(x)+\dim(u)$ inputs (scalar outputs)
  - Standard Method: Q is a Deep Neural Network (DNN)
    - QT-Opt says runtime non-linear opt of $Q^*(x,u)$
    - e.g. deepmind robot arm farm

- Actor-Critic

policy gradient $\Delta\alpha = -\eta \left[J(x+\beta) - b \right] \left[\frac{\partial}{\partial \alpha} \log f_{x|\alpha}(x|\alpha) \right]^T$ ← similar from last lecture: $[g(\alpha+\beta)-b]\beta$

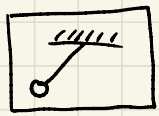
$J(x+\beta)$: one time experiment; b: baseline; $\frac{\partial}{\partial \alpha}\log f_{x|\alpha}(x|\alpha)$: policy gradient from REINFORCE; $\frac{\partial}{\partial \alpha}\pi$

Idea: use $J^\pi(x)$ as the baseline ← expected value (better baseline than running an extra trial)

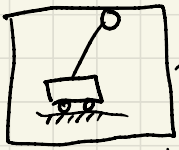
"Critic"

- Two time scale - Controller should change fast but Cost fn change slowly
- if reset controller (policy) but keep Value function (cost-to-go), controller reconverges quickly
  - indicates importance of learning cost-to-go & not just control policy (Policy gradient)
- Russ thinks this method of learning value fn is most promising
  - Eg. AlphaGo did this (though they also used a model & full state info)
    - use Monte-Carlo tree search (stochastic MPC) where terminal cost is cost-to-go learned by the value function

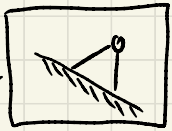
- Course Summary:



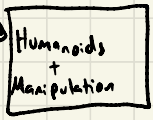
- Dynamic Programming/ Value iteration
- Simple non-linear system w/ few DoF



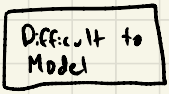
- Traj Opt
- LQR
- SoS
- LQR-Trees



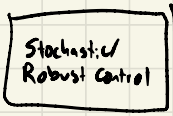
- Hybrid
- Limit Cycles



- Higher DoF
- Sample Motion Planning
- MI-Convex Programming



RL



- Polytopic Uncertainty