Lecture 11: Trajectory Optimization

Recap

- Dynamic Programming solve $J^*(x)$
  - on a mesh
    - neural value iteration (less guarantees in terms of convergence)
- LQR
  - scales to arbitrarily high dimensions
  - restricted to linear systems
- SOS/Lyapunov methods

Basic Trajectory formulation

$\dot{x} = f(x,u)$ ← dynamical system

Instead of saying our decision variables are somehow like a cost-to-go everywhere, we are going to restrict ourselves to just a trajectory

$$\min_{x(\cdot),u(\cdot)} \int_0^{t_f} l(x(t),u(t))\,dt$$

s.t. $\dot{x} = f(x,u), \forall t$ (dynamic constraints)

(u is a cont. traj, x is a cont. traj; some class of cont. curves which satisfy all eqns.)

+ additional constraints

for one initial cond. → $x(0) = x_0$

Linear discrete-time

$$\min_{x[\cdot],u[\cdot]} \sum_{n=0}^{N-1} l(x[n],u[n])$$

Convex cost fun (like linear in the cost fun or a quadratic in the cost fun)

($x[\cdot],u[\cdot]$: finite set of decision variables which represent x & u at particular time)

($n=0$ to N: some finite horizon)

s.t. $x[n+1] = Ax[n] + Bu[n]$

$x[0] = x_0$

+ additional constraints

$u_{min} \le u[n] < u_{max} \; \forall n$ (input limits are all linear constraints)

nice thing about the linear dynamics is that the dynamics constrain is a linear constraint in the optimization problem

Decision variable $x[1], \ldots, x[N]$

(Note: $x[0]$ is not a decision variable since it's fixed)

$u[0], \ldots, u[N-1]$

"Direct transcription"

So, if we are willing to choose some convex cost function like linear in the cost function or a quadratic in the cost function, then we are in the sweet where even though we are doing trajectory optimization now, we can still have a convex optimization problem.

Convex costs

$$l(x,u) = x^T Q x + u^T R u, \quad Q \succeq 0, \quad R \succ 0$$

$\Rightarrow$ Quadratic program

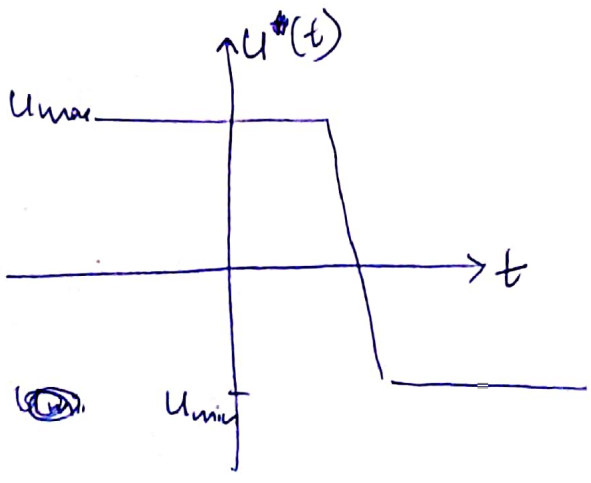
(quadratic cost and linear constraints)

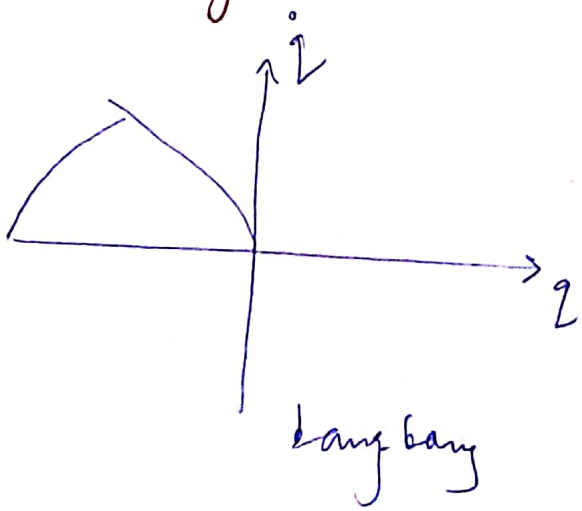
(then we are in the realm of quadratic programming)

Note – If we want to solve the minimum time problem we can actually solve many convex optimization problems, just keep increasing t_f until the first one the succeeds, that's our minimum time.

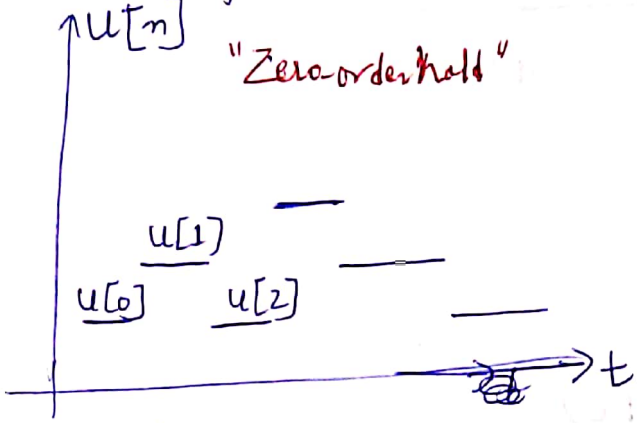
Discrete time approximation $\rightarrow x[n+1] = x[n] + \Delta t\,(Ax[n] + Bu[n])$

"Euler" Integration





Discrete time optimization



At each time, we are evaluating the dynamics as if u is fixed.

A first-order hold would be linear interpolation.

Linear discrete-time traj. optimization

Looks a lot like LQR but doing something LQR couldn't do. We could put input limits, ~~and~~ state constraints. In LQR, it's hard to solve for input limits and state constraints for all x.

Alternative "Direct Shooting" transcription

Idea - Solve away $x[n]$

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

$$\Downarrow$$

Given $x[0], u[0], \ldots, u[N-1]$

$$x[n] = A^n x[0] + \sum_{k=0}^{n-1} A^{n-k-1} B u[k]$$

$$\min_{u[\cdot]} \sum_{n=0}^{N} x^T[n] Q x[n] + u^T[n] R u[n]$$

$$\text{s.t. } x[0] = x_0$$

Compare Shooting Vs Direct transcription

- Shooting has less decision variables. — (good).
- Numerics can be worse
  A^n (if n is big)
- Sparsity missing.

Model predictive Control (MPC)

Direct shooting doesnot give us feedback controller out of the box. If we just solve that once, it only tells us how to go along from one initial condition. But if we start executing the trajectory and there is a disturbance that knocks us of the trajectory then the initial solution doesnot tell us what to do.

LQR was better in that sense. It gave us a whole policy. Shooting transcription gave us just a trajectory.

MPC — the idea that if we can solve the optimization reliably and fast enough then just solve it on every time step. Otherwise we find ourself go ahead & solve the optimization problem (like Direct Shooting) and that gives us a feedback controller.

→ MPC solves trajectory optimization on every time step.

→ If we have an optimization trajectory and we want a policy (if we can trust trajectory optimization) then it can give us policy.

Explicit model predictive control

Explicit model predictive control — That tries to understand what the solution would be for all states, try to turn that into a policy that we pre-compute so that we don't have to solve trajectory optimization online and that field of explicit model predictive control is hugely valuable but more for the theory it gave us than the practice because basically it tells us that the optimal policy the cost-to-go gets complicated really fast.

→ Explicit MPC is trying to solve for the optimal trajectory from all initial conditions for whatever cost functions.

Wouldn't it have been nice if Δt (time step) had been a decision variable in linear discrete-time case (direct transcription)?

$$x[n+1] = x[n] + h\left(Ax[n] + Bu[n]\right)$$

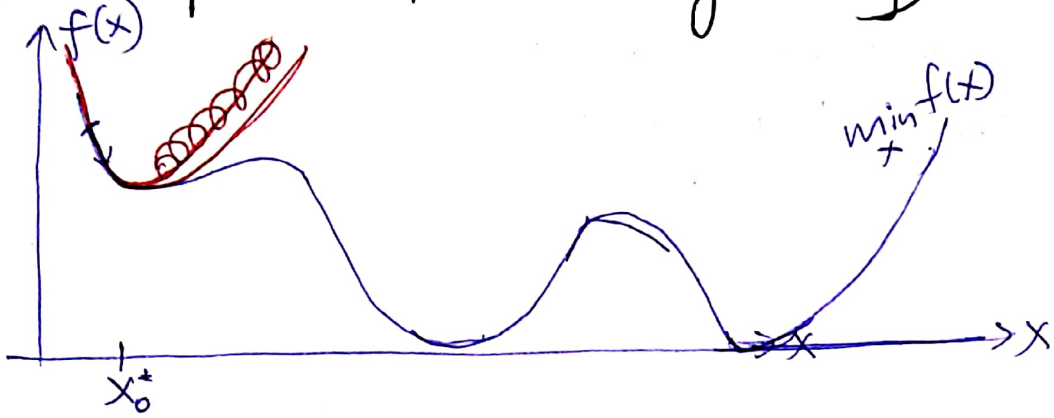
time step

bilinear

If we make 'h' a decision variable also, then what used to be linear constraint becomes a bilinear constraint and in general, that's not a convex function.

Non-linear Optimization

SNOPT (Sequential Quadratic Programming)



Makes a local quadratic approximation of the function and will use the 2nd order update to go directly to the minimum and then it will take a new quadratic approximation, jump to the minimum.

→ At each step, it take a quadratic approximation of the cost and a linear approximation of the constraints and it can solve the constraind optimization and then repeat.

Local minima & trajectory optimization

Issue
Sometimes it gets stuck to local minima

Scott Kindersma
- trajectory optimization
- Atlas
- MPC

Trajectory optimization for dynamic soaring
— Lukas Lao Beyer

Dexterity