Model Predictive Control

Monday, August 24, 2020 9:06 PM

Big Idea:

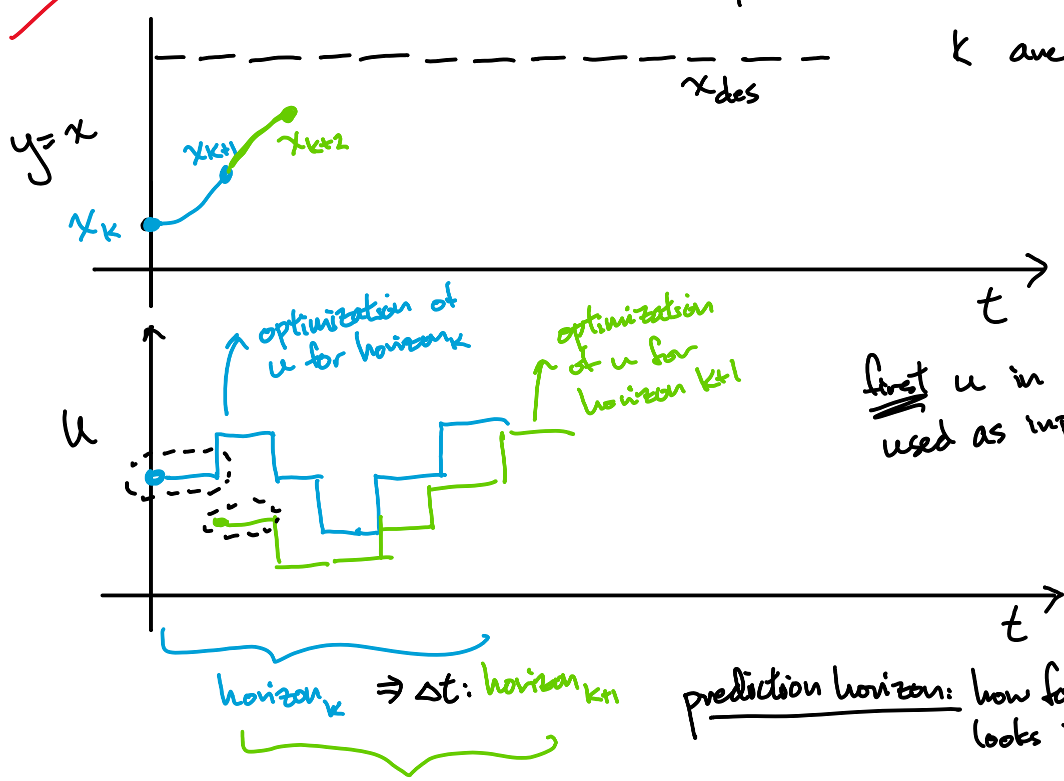
- optimizing control strategy for short window of time forward for system

$$\dot{x} = Ax + Bu \quad \text{(system)}$$
$$y = Cx + Du \quad \text{(output)}$$

CONCEPT OVERVIEW

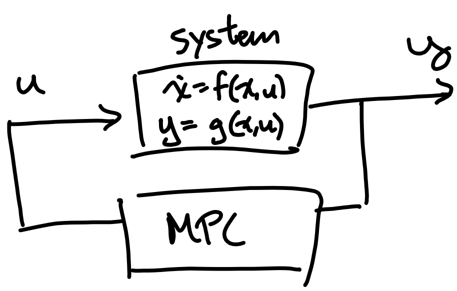
x: state $\quad u$: input

k are time steps



first u in horizon used as input

prediction horizon: how far controller looks into future



Advantages:

- can accept constraints
  - hard/soft
  - on state or inputs
- can handle nonlin. sys.
  - local linearization
- optimal
- robustness

⚠ expensive computationally (online), requires fast hardware

Important Remarks:

- Known optimal us for linear systems
  - nonlin. sys often linearized about some state or simplified
- w/ faster hardware, methods to find u for nonlin. systems directly exist
- b/c u is optimized over each timestep, much more robust to disturbances → simply replan u s.t. $x \to x_{des}$

↳ does not require invariant systems,

$\dot{x}$ can equal $\dot{x} = A(\mu)x + B(\mu)u$

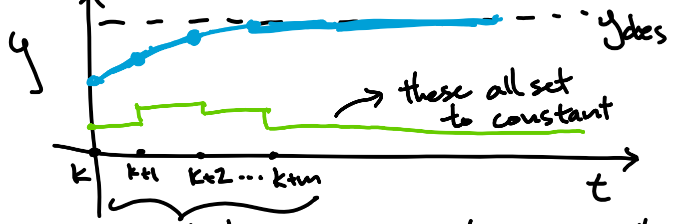
MPC DESIGN PARAMETERS

- sample time, T_s
- prediction horizon
- control horizon
- constraints
- weights

- <u>sample time:</u> time steps b/w control inputs
  - too large ⇒ slow response
  - too small ⇒ fast response, but computationally expensive
  - heuristic – roughly 10-20 samples in rise time of open loop system response

- <u>prediction horizon:</u> how far ahead to plan inputs
  - long: comp. time, unneeded
  - short: not robust to changes in desired state
  - recommended to have ~20-30 samples

- <u>control horizon:</u> number of time steps w/ "free" control input to be used



control horizon ⇒ optimization over these m inputs

! why not just use control horizon of one step?
- probably not best u, b/c state evolution unknown for longer times

! why not have control horizon = prediction horizon?
- comp. cost, and usually first few us have greatest effect on x.
- ~10-20% of prediction horizon w/ min 2-3 steps

- constraints: soft (violatable) + hard (inviolable)
  - generally, don't want hard constraints on input and output, b/c they can conflict
  - want to avoid hard constraints on inputs, $\dot{\text{inputs}}$, and on outputs

- weighting: we want "smooth" control moves (small changes from $u_k \to u_{k+1}$) and faithful tracking ($x \to x_{des}$)
  - for MIMO (multi-input/output) systems, relative weights can be assigned to prioritize certain inputs and outputs

NONLINEARITIES

Linear sys
Linear constraints
Quad. cost

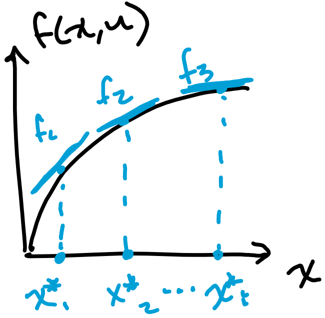
} ⇒ "Convex optimization problem" w/ global minimum :)

Linear MPC handles this well

- what about nonlinear systems? $\dot{x} = f(x, u)$

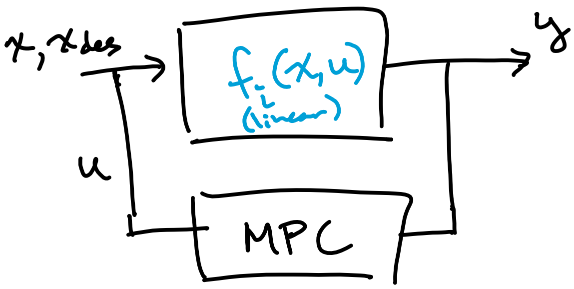
Method 1 Adaptive MPC:

- linearize $f(x,u)$ about series of points of interest



x_i^* at runtime

- use linearly approximated dynamics in MPC model as x changes ⇒ "adapt" the current plant based on current location



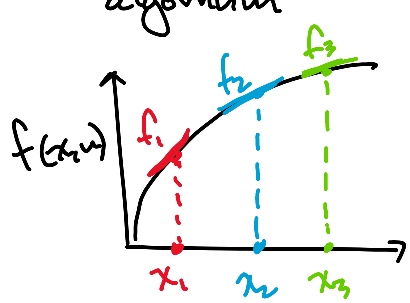
! Assumes # of states and constraints don't change as we change state

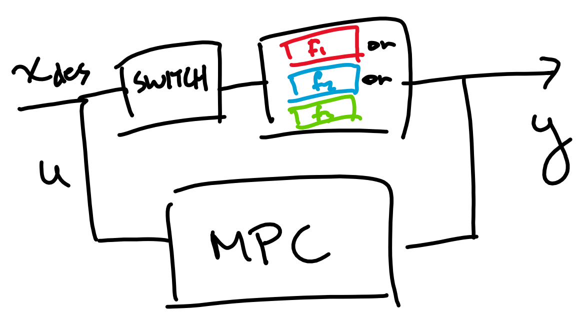
what if they do?

Method 2

Gain-Scheduled MPC:

- offline, make linearized models of dynamics at various operating points, can have unique #s of states/constraints
- stored in memory
- switch b/w "active" dynamics model according to algorithm





! Adaptive if linear $f^*(x^*,u)$ can be found at runtime, consistent xs and constraints

G-S if unique constraints and xs, but uses much more memory to store models offline

- what about nonlin. dynamics, constraints, <u>and</u> cost?

<u>screwed.</u> Nonlinear MPC available, but v. expensive, many local minima.

Optimization

So how do MPCs find the optimal us over a control horizon?

Typically, we want to define cost s.t. global minima exist, i.e., quadratic

$$\boxed{J = \hat{x}^T Q \hat{x} + u^T R u}$$

cost $\qquad \hat{x} = x - x_{des}$ ← and this

choose these

$\left.\begin{matrix} Q \\ R \end{matrix}\right\}$ weighting matrices for relative importance of variables in x, u.

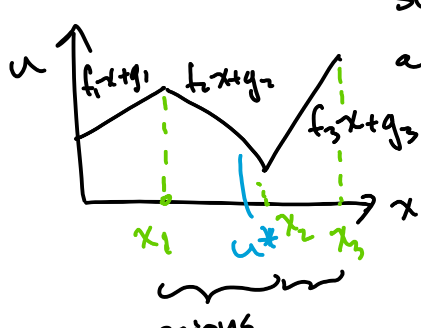
- this is a solvable problem for continuous and discrete dynamics.
- the optimizer will find a u^* s.t. J is minimized

$\Downarrow$

! Many <u>many</u> details as to why this works, but beyond basic summary. Possibly covered more in depth in optimization topic

PERFORMANCE

<u>Explicit MPC:</u> if we presolve for optimal us in the state space, gives piecewise linear affine functions



- controller just needs to find region x is in and evaluate for u^*
- "premade" library of solutions

regions

what if storage is limited, or there are many regions?

Suboptimal Solutions:

- worst case: finding u^* online at a time step takes longer than the whole step

- idea: cap the # of iterations taken to find a solution for u_k^*
  - ↳ use that sub-optimal u anyways, move on.
  - ↳ like trying to finish many HW assignments all due tomorrow

    $y \to$ grade average $x \to$ "smartness"

    $u \to$ studying