Lagrangian Mechanics

Lagrangian equation: $L = T - V$ → similar to balance $F - ma = 0$

↳ T is kinetic energy
↳ V is potential energy

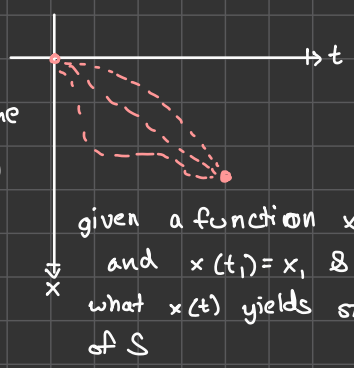
↳ T + V is total energy
↳ $T = T(q, \dot{q})$ and $V = V(q)$

"the state of motion in a particular state"

The principle of stationary action

$$S = \int_{t_1}^{t_2} L(x, \dot{x}, t)\, dt$$

S: action; L: relies on x(t); dt: ↳ time



given a function $x(t)$ and $x(t_1) = x_1$ & $x(t_2) = x_2$
what $x(t)$ yields stationary value of S

Euler-Lagrange Formulation

An n-dof system has n equations of motion (EoM)

The EoM for coordinate q_i with τ_i being a generalized F:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \tau_i$$

q_i (any variable that describes configuration)

↳* can = 0 when no non-conservative forces are acting on the object (forces don't depend on path taken)

example: $\Sigma F = m\ddot{x} = f - mg$

$x \to q$

$KE = \frac{1}{2}m\dot{x}^2$, $PE = mgx$ → $L = \frac{1}{2}m\dot{x}^2 - mgx$

$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) = \frac{d}{dt}(m\dot{x}) = m\ddot{x}$

$\frac{\partial L}{\partial q} = -mg$

$m\ddot{x} - (-mg) = \tau$

$\tau_i = \sum_{j=1}^{k} f_j^T \frac{\partial r_j}{\partial q_i}$ for k points r_j and k forces f_j

↳ $F^T J = J^T F \to \tau = J^T F$

Constraints

Example: disk rolling without slipping
Using Lagrangian makes dealing with these easier

Pick N-k variables q_i for a system with k constraints

Holonomic: only dependant on q_i not $\dot{q}_i$
↳ Lagrangian handles this easily

Example Systems

A particle rolling in a frictionless cone with angle α

Configuration variables: r, θ, z

$KE = \frac{1}{2}m(\dot{r}^2 + r^2\dot{\theta}^2 + \dot{z}^2)$

$PE = mgz$

Due to the cone constraint, we can do $N - k = 3 - 1 = 2$ and remove a variable as $z = r\cot(\alpha)$ and $\dot{z} = \dot{r}\cot(\alpha)$

$$L = \frac{1}{2}m\left[\dot{r}^2 + r^2\dot{\theta}^2 + \dot{r}^2\cot(\alpha)^2\right] - mgr\cot(\alpha)$$

Using the Euler-Lagrange formula we can derive 2 EoMs

For r: $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{r}}\right) - \frac{\partial L}{\partial r} = 0 \to \frac{d}{dt}\left[m(1+\cot^2\alpha)\dot{r}\right] - mr\dot{\theta}^2 + mg\cot\alpha$

$(1+\cot^2\alpha)\underline{\ddot{r}} - r\dot{\theta}^2 + g\cot\alpha = 0$

For θ: $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0 \to \frac{d}{dt}(mr^2\dot{\theta})$

$2\dot{r}r\dot{\theta} + r^2\underline{\ddot{\theta}} = 0$

Walking Robots

Fundamentals

Basic Models of Contact

this is the spring-dampner model where a spring pushes the leg out of the ground plane

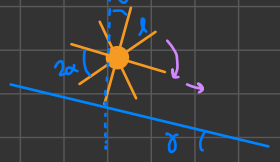
*good for simulation but bad for analysis (large k)

this is the pin joint model where you add and release joints as make/break contact

*hybrid model of rigid contact because discont

- at the moment of impact there is a new constraint added pos(foot) = constant
- impulsive collision event (inelastic)

Rimless Wheel

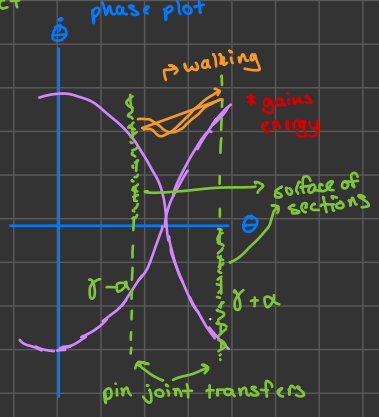


the simplest model for walking is a wheel without the rim

the PE converts to KE down the ramp with a rolling-like motion but with contact collisions "making and breaking contact"

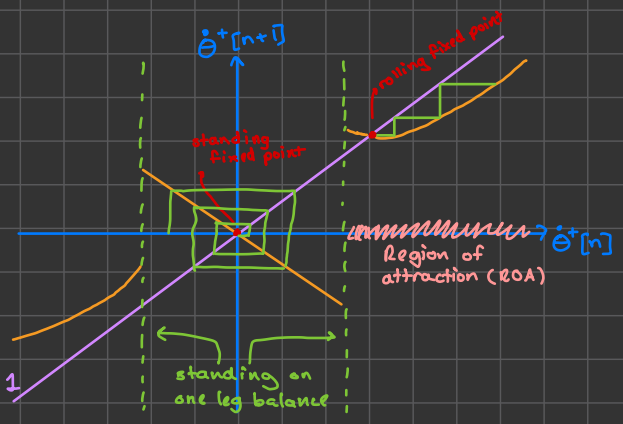
this is just an inverted pendulum →

interesting behavior is when you start it fast, it slows down to stable speed and when you start slow, it speeds up



Angular momentum is conserved around the point of impact

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



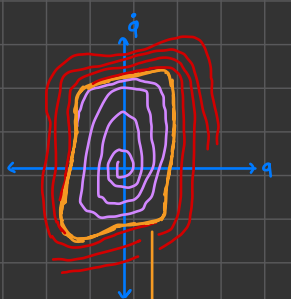
Poincaré Map for Rimless Wheel

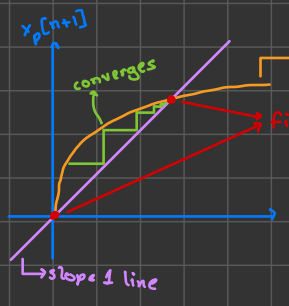
What are Poincaré Maps?

In a state space of size n we can take a n-1 dimentional surface of section that the flow is transverse to. Poincaré maps this:

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

↳ is the nth crossing of surface



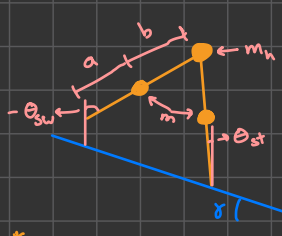


→ poincaré return map for a Van der Pol Oscillator

fixed point → $x_p[n+1] = x_p[n]$

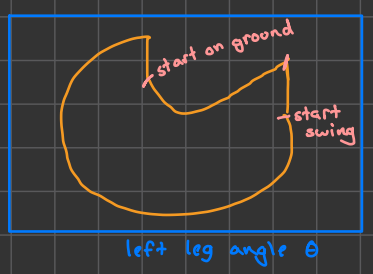
if the |eigenvalues| of $\frac{\partial P}{\partial x_p}$ are ≤ 1 then it will converge because its like saying the derivative on the right figure is ≤ 1

Compass Gait



this is the rimless wheel with only 2 spokes and added point masses to the legs

* the hip is now a pin joint itself



Hybrid Trajectory Opt

chapter 17

Find limit cycles

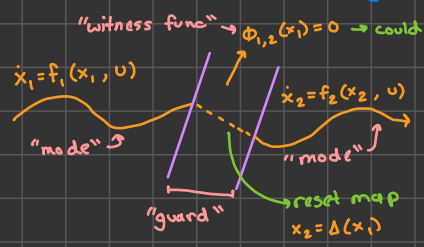
find $x(), T$ s.t. $\dot{x} = f(x)$, $x(0) = x(t)$ an initial guess of a clockwise circle works and converges very fast

for the case of a rimless wheel dynamics model:

find $x(\cdot), T$ s.t.
- $\dot{x} = f(x)$ → dynamics
- $\theta(0) = \gamma - \alpha$ → initial new foot
- $\theta(T) = \gamma + \alpha$ → foot switch
- $\dot{\theta}(0) = \cos(2\alpha)\,\dot{\theta}(T)$ → collision loss

produces stable solution to passive walkers

Autonomous Hybrid System



this shows a system evolving until a discrete event changes state and the evolving from the new state x_2

Examples of discrete events. First heel contact then toe contact.

Trajectory optimization formulation:

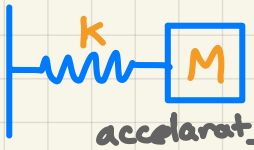
find $x_k[\cdot], h_k$ s.t.
- $x_0[0] = x_0$
- $\forall k$ $\Phi_{k,k+1}(x_k[N_k]) = 0$ → guard happens
- $x_{k+1}[0] = \Delta_{i,j}(x_k[N_k])$ → Δstate before/after guard
- $\Phi_{k,k'}(x_k[n_k]) > 0 \quad \forall n_k, \forall k'$ → always +
- ...

State Space Control

-Representing dynamic systems with differential equations

Definitions

- How a system is changing = f(current state)



$M\ddot{p}(t) = -k\,p(t)$

Simple Spring-Mass System

acceleration → $\ddot{p}(t) = \frac{-k}{M}p(t)$ ← position

$\dot{x} = f(state, inputs)$

Equations

how internal states connect ←

which states they affect ↑

State equation ⇒ $\dot{x}(t) = Ax(t) + Bu(t)$

how system changes ← $\dot{x}(t)$

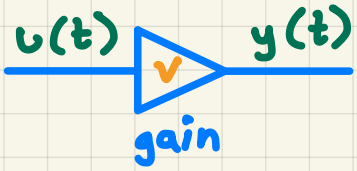
A ↓ state vector

B ↓ inputs

how states combine to get outputs ←

allows inputs to feedforward to output ↑

Output equation ⇒ $y(t) = Cx(t) + Du(t)$



$y(t) = \boxed{value} \cdot \boxed{u(t)}$

More

In spring system, state = position, velocity

PE ↓ KE ↓

future of the system ← $\dot{x} = A \cdot x$

A ↓ how energy transfers given state

- a matrix A defines a linear transfromation of some n-dimention space
  ↳ how the system behaves under no external force

- solving $\dot{x} = Ax$ as a linear differential equation results in $x(t) = e^{At}x_0$ where A is all $e^{\lambda t}$ terms superimposed
  ↳ $\frac{d}{dt}\vec{v}(t) = A \cdot \vec{v}(t)$
  ($\frac{d}{dt}\vec{v}(t)$: state; A: dynamics matrix)

-Jacobians are used to see how a state will change in a matrix (like derivative)

Eigenthings

- $e^{At}x_0$ is the short form of defining a state vector based on dynamics matrix A at time t
  ↳ if is a taylor series expanding to $x_0(A^0 + A^1 + \frac{1}{2}A^2 + \ldots + \frac{1}{n!}A^n)$

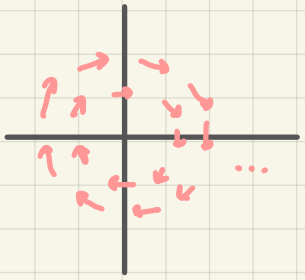
- the eigenvalues of the system describe the constant direction scalar mults (λ)
  ↳ if $\lambda < 0$, $e^{\lambda t}$ approaches 0 (stable)
  $\lambda = 0$, $e^{\lambda t}$ is constant (semi stable)
  $\lambda > 0$, $e^{\lambda t}$ expands to ∞ (unstable)

- $Ax = \lambda x$
  $A = \lambda I$
  $\det(A - \lambda I) = 0$
  $\det\begin{bmatrix} a-\lambda & b \\ c & d-\lambda \end{bmatrix} = 0$ finding an eigenvalue
  *can be any size

- control law sets $u = K(r - x)$ where K is gains and $r - x$ is error from setpoint in state
  ↳ $\dot{x} = Ax + BK(r-x)$
  $= Ax + BKr - BKx$
  $= x(A - BK) + BKr$

  $y = Cx + Du$
  $= Cx + DK(r-x)$
  $= x(C - DK) + DKr$

- Vector fields define a state space



$e^{At}x_0 = e^{At} \cdot \begin{bmatrix} x \\ 0 \end{bmatrix}$
$dx = (v_x \cos\theta(t) - v_y \sin\theta(t)) \cdot dt$
$dy = (v_x \sin\theta(t) + v_y \cos\theta(t)) \cdot dt$
$\begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) \\ \sin\theta(t) & \cos\theta(t) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \cdot dt$
$\Delta\theta = t \cdot \omega$
$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) & 0 \\ \sin\theta(t) & \cos\theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} dt$
bad estimate in case of changing θ

better estimate:
$dx = v_x$

Summary

- a mat A defines linear transform
- eigenvectors of A with a value less than 0 will approach stable origin
- with $e^{At}x_0$ we can get a new vector at time t given initial vector

Constrained nonlinear optimization

Fundamentals

What is an nlp?

$$\min_{x \in \mathbb{R}^n} f(x)$$
$$h_i(x) = 0 \text{ for } i \in \mathcal{E}$$
$$g_i(x) \leq 0 \text{ for } i \in \mathcal{I}$$

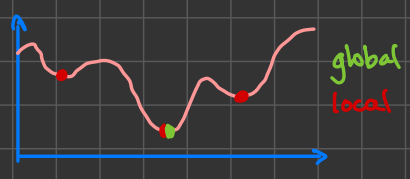
f is the function of x to minimize
x is a vector of controllable real numbers
h_i is an element in a list of $\mathcal{E}$ equality constraints
g_i is an element in a list of $\mathcal{I}$ inequality constraints

the constraints define the "feasible region" where the solution can be found

What is a minimum?

This is at a solution x^* for which $f(x)$ is at a local or global minima. The global minimum is just the best local minimum but is often very difficult to get.

↳ for convex problems, the only local min is also a global min



$\nabla f(x) = 0$ and $\nabla^2 f(x) > 0$ at x^*

What is a gradient and what is descent?

the gradient $\nabla f(x)|_a$ tells us the direction of steepest ascent at point a

Decent is moving down a cost $f(x)$ where $f_{k+1} < f_k$

Methodology

Newton's direction

Step vector p that minimizes the second order taylor for $f(x_k + p)$

$$\approx f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T \nabla^2 f(x_k) p \overset{\text{def}}{=} m_k(p)$$

which has a $\nabla = 0$ at $p = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$

!! Note: this assumes ∇^2 is > 0 and $\|p\|$ is small enough that taylor is accurate

$\nabla f_k^T p_k^N < 0$ to ensure the Newton's direction is descent

Trust Regions

$$\min_{p} f_k + p \nabla f_k \quad \text{subproblem in trust region}$$
$$\|p\| < \Delta_k$$

$p \rightarrow$ step vector
p is $<$ trust region radius Δ_k

$p = -\Delta_k \cdot \frac{\nabla f_k}{\|\nabla f_k\|}$ is steepest descent in trust region

Quasi-Newton

↳ alternate to newton that doesn't require hessian

uses the fact that every step's $\Delta \nabla f(\cdot)$ contains info about ∇^2 in a discrete manner (B is approx Hess)

$$B_{k+1} \cdot (x_{k+1} - x_k) = (\nabla f_{k+1} - \nabla f_k)^*$$

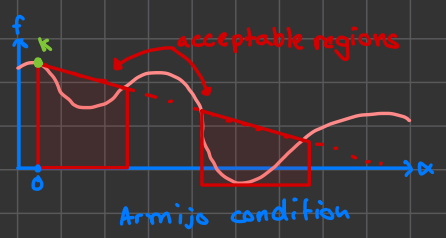
*SR1 or BFGS formula is better for actual use

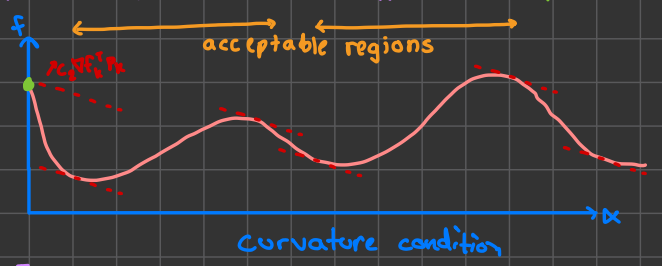
therefore, $p = -B_k^{-1} \nabla f_k$

Line Search methods

↳ how to choose a step size α

Armijo condition: $f(x_k + \alpha p_k) < f(x_k) + c_1 \alpha \nabla f_k^T p_k$ where c_1 is const





Curvature condition: $\nabla f(x_k + \alpha p_k)^T p_k \geq c_2 \nabla f_k^T p_k$ $\quad 0 < c_1 < c_2 < 1$

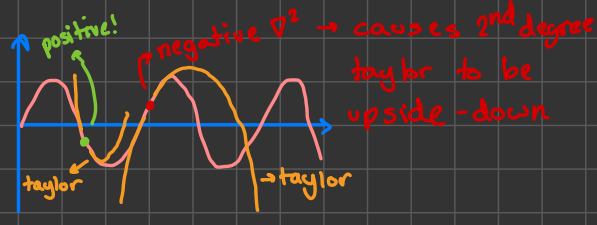
Together, armijo + curvature condition are known as the "Wolfe" conditions

Techniques

Newtons method + hessian modify

Resolves the issue where the Hessian is not positive-definite causing direction to not be descent.

☆ Determine if matrix is positive using Cholesky factorization (can also be done with eigen-values)



→ causes 2nd degree taylor to be upside-down

The hessian is modified by adding a positive diagonal matrix $\begin{bmatrix} x_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & x_n \end{bmatrix}$

general method process:

for k in $0, 1, 2, \ldots$

→ Factorize $B_k + E_k$ and choose E_k until sufficiently positive-definite

→ solve $B_k p_k = -\nabla f(x_k)$

→ $x_{k+1} = x_k + \alpha p_k$ satisying wolfe or other conditions

E_k can be $(k+1)(\beta)I$ where I is the identity matrix

read ☆ above

The spectral decomposition $Q \Lambda Q^T$ can be used to modify the eigenvalues stored in the diagonal of Λ according to [$\lambda < \delta \rightarrow$ replace with δ, otherwise remain same] mathematicaly this can be written as:

$Q \Lambda Q^T + \Delta A$ where $\Delta A = Q \text{diag}(\tau_i) Q^T$ with

$$\tau_i = \begin{cases} 0, & \lambda_i \geq \delta \\ \delta - \lambda_i, & \lambda_i < \delta \end{cases}$$

☆ Cholesky with multiple of I process: ☆

```
β = 10^-3
if (min diag in hess > 0): τ = 0
else: τ = -(min diag) + β
for (k = 0,1,2...):
    if (cholesky ü): exit
    else:
        τ = max (10 τ, β)
        hess += I · τ
```

Step-length selection

Manifolds

Smooth Manifolds

What are manifolds

Spaces that locally look Euclidean. Examples include smooth curves like circles and higher dimensional figures like a sphere or torus.
↳ eg, an n-dimensional hypersphere, etc

Trajectory Optimization

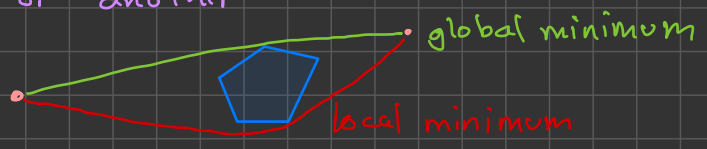
Fundementals

Trajectory optimization is usually a nonlinear optimization problem to be solved by NLP methods

This can be use for Model Predictive Control (MPC) or full movement problems

Local Minima

example of local minimum in traj opt is going around an obstacle one way instead of another



Methods

Direct Transcription

This method involves adding state array x[] as a decision variable with inputs u[] resulting in the following form:

$$\min_{x[\cdot],u[\cdot]} \; J(x[\,],u[\,])$$

- other constraints

for linear systems this is

$$x_{n+1} = Ax_n + Bu_n$$

and otherwise

$$x_{n+1} = x_n + \int_{t(n)}^{t(n+1)} f(x_n, u_n)\,dt$$

Direct Shooting

Remove x[.] from decision variables since you can solve for it given u.

$x_2 = A[Ax_0 + Bu_0] + Bu_1$ and keep expanding

This works well enough for linear systems but not recommended for NLP because of coeffs buildup's effect on numerical issues

$$x_n = A^n x[0] + \sum_{i=0}^{n-1} A^{n-1-i} Bu[i]$$

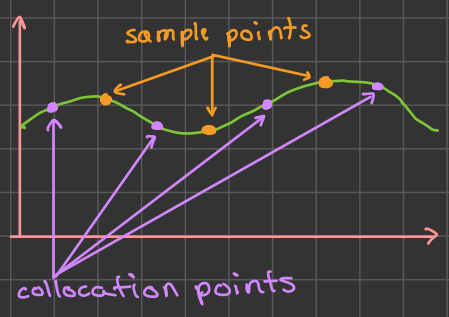
"I tend to favor the other ones" - Russ Tedrake

Direct Collocation

↳ Russ Tedrake's favorite

The state trajectory is represented as a cubic polynomial and input as a first-order polynomial. This also makes it always non-convex

Note that x_k, $\dot{x}_k$, x_{k+1}, $\dot{x}_{k+1}$ together define the cubic polynomial from t_k to t_{k+1}



we can sample points easily to add constraints

in direct collocation we have to add a derivative constraint at collocation points $t_{c,k}$

assuming the collocation points are at time segment midpoints, means $t_{c,k} = \frac{1}{2}(t_k + t_{k+1})$ where the derivative comes from interpolating the cubic spline to its midpoint

Faster!

Custom Solvers

custom problem-specific solvers can dramatically outperform general purpose solvers like IPOPT. A popular way to do this is transforming the constrained problem into an unconstrained one using penalty methods (usually based on augmented Lagrangian)

Handmade Gradients

you can often outperform autodiff by writing smaller, better expressions for gradients than autodiff's general methods

For direct trascription and collocation $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial u}, \frac{\partial l}{\partial x}, \frac{\partial l}{\partial u}$ where l is cost and f is the dynamics function

For direct shooting since its nested, we can use chain rule through $x_n = f(f(f(\ldots f(x_0,u_0),u_1),u_2\ldots)$ (backprop)

Constraints

Cost vs Constraint

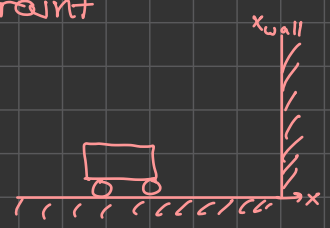
"I have to be outside the tree vs I enjoy being outside the tree" - drake
Simple cost functions are best (for example u^2 is very often good)

Solvers will do stuff like augmented lagrangian to put constraints in cost behind the scenes though (these look very different)

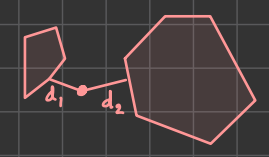
So generally, put conditions you want true in constraints and what you're iterating to get better in cost (eg time = cost and collision = constraint)

Minimum Distance Constraint

Nice linear constraint: $x \leq x_{wall}$



For polygonal constrainsts, we use a signed distance function
For this case we'd like $\min(d_1, d_2)$ to be greater than min.
But this is bad for many bodies and kink when switching which is closest
We solve this by ignoring obstacles > a certain safe maximum view distance.



This constraint is 0 in good condition, linear in bad and smooth transition between the 2

Stability

Linearing around a trajectory

time-varying linearizations can stay accurate as time passes because the A and B matrices are a function of time

$$\dot{x} \approx f(x_0, u_0) + \left.\frac{\partial f}{\partial x}\right|_{\substack{x_0 \\ u_0}} (x - x_0) + \left.\frac{\partial f}{\partial u}\right|_{\substack{x_0 \\ u_0}} (u - u_0)$$ ← this is linearized around the point (x_0, u_0)

to make it time variant, we write $x_0(t)$ and $u_0(t)$

$\bar{x} = x - x_0(t)$ $\quad \dot{\bar{x}} = \dot{x} - \dot{x}_0(t) \longrightarrow \dot{\bar{x}} = A(t)\bar{x} + B(t)\bar{u}$
$\bar{u} = u - u_0(t)$

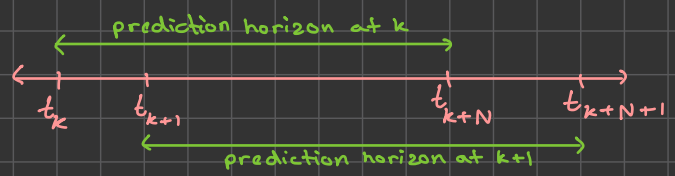
Model Predictive Control (MPC)

Fundamentals

What is it?

If we can do trajectory optimization fast enough, we can use it as the feedback controller/policy (convex)

Involves optimizing over the next N timesteps and since it keeps moving this is called "receding horizon"



Guaranteeing Stability

recursive stability is the promise that if a solution is found at n then one will also be found at $n+1$

simplest way to do this is stabilizing around a fixed point (x^*,

At point k we optimize $x_k \dots x_{k+N}$ and $u_k \dots u_{k+N-1}$ and constrain $x_{k+N} = x^*$ because then at $k+1$ we can

Figure this out!!

Nonlinear Dynamics

dynamics are usually nonlinear so that can ruin the convex nature of a MPC that we rely on for fast solve times. we solve this by linearizing the model at each time step to reach dynamics:

$\dot{x} = Ax + Bu$ ← all expressions in matmul are linear!

Problem Formulation

the cost function for a trajectory tracking MPC can be:

$$J_{MPC} = \sum_{i=0}^{N} (x_{err_i}^2)$$

Optimal Control

Fundamentals

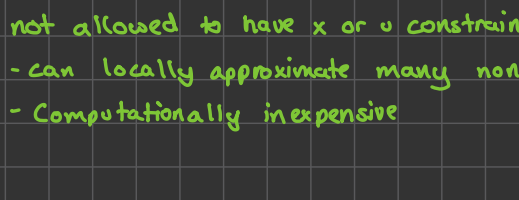
Deterministic Optimal Control

Continuous time optimal control optimization problem:

$$\min_{\substack{x(t)\\u(t)}} \; J(x(t),u(t)) = \int_{t_0}^{t_f} \ell(x(t),u(t))\,dt + \ell_f(x(t_f))$$

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

- $J(x(t),u(t))$: cost function
- $\ell(x(t),u(t))$: stage cost
- $\ell_f(x(t_f))$: terminal cost
  - could be high if goal not reached
- $x(t), u(t)$: state + input trajectories
- $\dot{x}(t) = f(x(t),u(t))$: dynamics constraint



$u(t) = \lim_{N\to\infty} u_{1:N}$ as samples goes to infinity

samples called knot points

Pontryagin's Minimum Principle

First order necessary conditions for deterministic optimal control problem:

Discrete time optimal control opt problem

$$\min_{\substack{x_{1:N}\\u_{1:N}}} \; J(x_{1:N},u_{1:N}) = \sum_{k=1}^{N-1} \ell(x_k,u_k) + \ell_f(x_N)$$

$$\text{s.t. } x_{n+1} = f(x_n,u_n) \quad \forall n \in [0,N)$$

$$\dots$$

Discrete → Continuous: integrators
Continuous → Discrete: interpolation

LQR

Linear Quadratic Regulator

Problem

$$\min_{\substack{x_{1:N}\\u_{1:N}}} \; J = \sum_{n=1}^{N-1} \tfrac{1}{2} x_n^T Q_n x_n + \tfrac{1}{2} u_n^T R_n u_n + \tfrac{1}{2} x_N^T Q_N$$

$$\text{s.t. } x_{k+1} = A_n x_n + B_n u_n \qquad Q \succeq 0,\; R \succ 0$$

not allowed to have x or u constraints
- can locally approximate many nonlinear problems
- Computationally inexpensive

As a QP

Define $z = [u_1, x_1, u_2, x_2 \dots x_N]^T$ and $H = \begin{bmatrix} R_1 & & & & \\ & Q_2 & & & \\ & & R_2 & & \\ & & & Q_3 & \\ & & & & \ddots \\ & & & & & Q_N \end{bmatrix}$ such that $J = \frac{1}{2} z^T H z$

Define C and d:

$$\underbrace{\begin{bmatrix} B & -I & 0 & 0 & & \\ A & B & -I & 0 & & \\ & & \ddots & & & \\ & & & A_{n-1} & B_{n-1} & -I \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ \vdots \\ x_n \end{bmatrix}}_{z} = \underbrace{\begin{bmatrix} -Ax_1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{d}$$

→ each one is $Ax_n + Bu_n - Ax_{n+1} = 0$

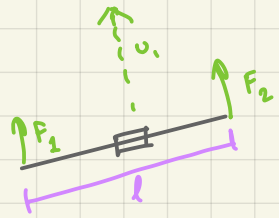
Now we write LQR as a QP: $\min_z \; \frac{1}{2} z^T H z \quad \text{s.t. } Cz = d$

The Lagrangian of this QP is $L(z,\lambda) = \frac{1}{2} z^T H z + \lambda(Cz - d)$

↳ KKT conditions: $\nabla_z L = Hz + C^T\lambda = 0$, $\nabla_\lambda L = Cz - d = 0$ $\Rightarrow \begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$

There's no iteration because one newton-step solves it (linear system)

①



$$\tau_1 = \frac{l}{2} \cdot F_1$$
$$\tau_2 = \frac{l}{2} \cdot F_2$$

$$\alpha = \frac{\tau_1 - \tau_2}{I} \qquad \alpha I = \frac{l}{2}(F_1 - F_2)$$

$$[x \;\; y \;\; \theta \;\; \dot{x} \;\; \dot{y} \;\; \dot{\theta}]^T = x$$

$$v_1 = F_1 + F_2$$
$$v_2 = \frac{l}{2}(F_1 - F_2)$$

$$F_1 = \frac{v_1}{2} + \frac{v_2}{l}$$
$$F_2 = \frac{v_1}{2} - \frac{v_2}{l}$$

$$[v_1 \;\; v_2]^T = v$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} \text{ aka } \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = A \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} + B \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$F_x = v_1 \cdot \cos(\theta + 90) = -v_1 \sin\theta$$
$$F_y = v_1 \cdot \sin(\theta + 90) = v_1 \cos\theta$$

$$\ddot{x} = F_x / m = \frac{-v_1 \sin\theta}{m}$$
$$\ddot{y} = F_y / m = \frac{v_1 \cos\theta}{m} - g$$
$$\ddot{\theta} = \frac{l(F_1 - F_2)}{2I} = \frac{v_2}{I}$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -\frac{\sin\theta}{m} & 0 & 0 \\ \frac{\cos\theta}{m} & 0 & -1 \\ 0 & \frac{1}{I} & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ g \end{bmatrix}$$

$$\ddot{x} = \ddot{x}_0 + \left.\frac{\partial \ddot{x}}{\partial \phi}\right|_{EQ} (\phi - \phi_0)$$

$$\delta\ddot{x} = -\frac{v_1 \cos\phi}{m} \cdot \delta\phi = -g \cdot \delta\phi = -9.8 \cdot \delta\phi$$

②

$$\ddot{y} = \ddot{y}_0 + \left.\frac{\partial \ddot{y}}{\partial g}\right|_{EQ} (g - g_0) + \left.\frac{\partial \ddot{y}}{\partial \phi}\right|_{EQ} (\phi - \phi_0) + \left.\frac{\partial \ddot{y}}{\partial v_1}\right|_{\phi=0} \cdot \delta v_1$$

$$\delta\ddot{y} = -\delta g + \frac{v_1 \cdot (-\sin\phi)}{m} \cdot \delta\phi + \left(\frac{\cos\phi}{m}(1)\right) \cdot \delta v_1 = -\delta g + \frac{\delta v_1}{m}$$

$$\ddot{\phi} = \ddot{\phi}_0 + \left.\frac{\partial \ddot{\phi}}{\partial v_2}\right|_{EQ} (v_2 - v_{2_0})$$

$$\delta\ddot{\phi} = \frac{\delta v_2}{I}$$

$$\begin{bmatrix} \delta\dot{x} \\ \delta\dot{y} \\ \delta\dot{\phi} \\ \delta\ddot{x} \\ \delta\ddot{y} \\ \delta\ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -9.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \delta x \\ \delta y \\ \delta\phi \\ \delta\dot{x} \\ \delta\dot{y} \\ \delta\dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m} & 0 & -1 \\ 0 & \frac{1}{I} & 0 \end{bmatrix} \cdot \begin{bmatrix} \delta v_1 \\ \delta v_2 \\ \delta g \end{bmatrix}$$

③

$$\begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \delta x \\ \delta y \\ \delta\phi \\ \delta\dot{x} \\ \delta\dot{y} \\ \delta\dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \delta v_1 \\ \delta v_2 \\ \delta g \end{bmatrix}$$

Jacobian linearization

$$\begin{bmatrix} \frac{\partial \ddot{x}}{\partial x} & \frac{\partial \ddot{x}}{\partial y} & \frac{\partial \ddot{x}}{\partial \phi} & \frac{\partial \ddot{x}}{\partial \dot{x}} & \cdots & \frac{\partial \ddot{x}}{\partial v_2} & \frac{\partial \ddot{x}}{g} \\ \frac{\partial \ddot{y}}{x} & \longleftarrow \cdots \longrightarrow & & & & & \frac{\partial \ddot{y}}{g} \\ \frac{\partial \ddot{\phi}}{x} & \longleftarrow \cdots \longrightarrow & & & & & \frac{\partial \ddot{\phi}}{g} \end{bmatrix}$$

split ↑ before inputs