ROBOT DYNAMIC REVIEW

Kinematics describe the motion without considering the forces involved.

HOMOGENEOUS TRANSFORMATION MATRIX represents the pose of a frame wrt another.

$$T_1^0 = \begin{bmatrix} R^0_1 & p \\ O_{1\times3} & 1 \end{bmatrix}$$

p is the 3x1 position vector

FORWARD KINEMATICS calculates the pose of the end-effector given the joint coordinates q by composing the trasformation matrices of each link.

$$T^0_n(q) = T^0_1(q)\, T^1_2(q_2) + \ldots + T^{n-1}_n(q_n)$$

DIFFERENTIAL KINEMATICS relates joint velocities to end-effector linear and angular velocities. **JACOBIAN MATRIX** maps joint velocities ($\dot{q}$) to the end-effector's cartesian velocity (Ve). The linear velocity Ve is composed of linear velocity $\dot{p}$ and angular velocity ω:

$$Ve = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = J(q)\dot{q}$$

STATICS relates forces and torques at the end-effector to the torques required at the joints. The joint torque (τ) required to balance an external wrench $w = [f^T, m^T]^T$ applied at the end-effector are calculated using the transpose of the Jacobian:

$$\tau = J^T(q)\, w$$

DYNAMICS describes relationship between forces acting on the robot and its resulting motion.

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau + J^T(q)\, w$$

MOTION CONTROL

Strategies to compute required joint torques (τ) to make the robot follow a desired trajectory.

JOINT SPACE CONTROL have as objective to make the robot's joint configuration $q(t)$ track a reference joint trajectory $q^{ref}(t)$. The tracking error is defined as $e = q^{ref} - q$

- PD + GRAVITY COMPENSATION : $\tau = K_p e + K_d \dot{e} + g(q)$ guarantees zero steady-error when desired velocity is zero.
- INVERSE DYNAMIC CONTROL : $\tau = M(q)(\ddot{q}^{ref} + K_p e + K_d \dot{e}) + h(q, \dot{q})$ make the sistem linear $\longrightarrow$ $\ddot{e} + K_d \dot{e} + K_p e = 0$

CARTESIAN SPACE CONTROL make end-effector pose $x(t)$ track a reference cartesian trajectory $x^{ref}(t)$.

- OPERATIONAL SPACE DYNAMICS : $\tau = J^T F$ calculates a cartesian force F needed to drive the end-effector. The force is then mapped back to joint torques.
- CONTROL LAW WITH REDUNDANCY : $\tau = J^T F + (I - J^T \bar{J}^T)\tau_0$ the null space of Jacobian can be used to perform secondary tasks. Control law is extended to null-space torque.

Quadratic Problem

1. QP-BASED REACTIVE CONTROL

JOINT SPACE CONTROL [standard PD control]

$M(q)\ddot{q} \to h(q,\dot{q}) = \tau \Rightarrow \begin{cases} \tau^d = M\ddot{q}^d + h \\ \ddot{q}^d = \ddot{q}^r + K_d(\dot{q}^r - \dot{q}) + K_p(q^r - q) \end{cases}$

↳ Dynamics equation

↗ Trasformare il problema di come controllare un robot in un problema di ottimizzazione (risolvibile in real-time)

Optimization problem that allow us to came at the same solution:

$(\tau^*, \ddot{q}^*) = \arg\min_{\tau, \ddot{q}} \frac{1}{2}\|\ddot{q} - \ddot{q}^d\|^2 = \frac{1}{2}\ddot{q}^T\ddot{q} - \ddot{q}^{dT}\ddot{q} + \frac{1}{2}\ddot{q}^{dT}\ddot{q}^d$ (QP)

↳ constant

↳ subject to $M\ddot{q} + h = \tau$ L'accelerazione che troviamo deve rispettare la fisica del robot

SOLUTION $\begin{cases} \tau^* = \tau^d = M\ddot{q}^d + h \\ \ddot{q}^* = \ddot{q}^d \end{cases}$

The advantage of this version of the problem is that we define:

- physical bound: $\tau^{min} \le \tau \le \tau^{max}$ coppia massima
- electric current limits: $i^{min} \le K_\tau \tau \le i^{max}$
- joint velocity limits: $\dot{q}^{min} \le \dot{q} + \Delta t\ddot{q} \le \dot{q}^{max}$

} linear constraints

Andiamo a limitare l'accelerazione

At the end, we have to solve:

$\min_{\tau, \ddot{q}} \frac{1}{2}\|\ddot{q} - \ddot{q}^d\|^2$ subject to $M\ddot{q} + h = \tau$ + constraints define above

↳ Use a solver to solve the optimization problem efficiently

VELOCITY CONSTRAINTS:
Since we can regulate only $\ddot{q}$, to manage velocity constraints we assume that $\ddot{q} = const$ during time step Δt: $\dot{q}(t+\Delta t) = \dot{q}(t) + \Delta t\ddot{q}$ so I rely on the next value of the velocity. (Agisco sull'accelerazione)

JOINT LIMITS: ↗ Applicare limiti ai giunti potrebbe richiedere accelerazioni enormi

$q(t+\Delta t) = q(t) + \Delta t\dot{q}(t) + \frac{\Delta t^2}{2}\ddot{q}$ $\quad q^{min} \le q + \Delta t\dot{q} + \frac{\Delta t^2}{2}\ddot{q} \le q^{max}$

↳ In practice doesn't work. (In EXTRA material there's the proof)

↳ This problem is not solvable with reactive control methods

Taxonomy of convex optimization problems

- LEAST SQUARE PROGRAM (LSP) → $Ax \le b$, $Ax = b$
  ↳ $\|Ax - b\|^2$
- QUADRATIC PROGRAMS (QP) → Linear constraints
  ↳ convex quadratic cost $\frac{1}{2}x^THx + g^Tx$, $H \succeq 0$

Can be solved efficiently ($\le 1ms$) by solvers.

LEAST SQUARE PROGRAMS

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|Ax-b\|^2 = \frac{1}{2}x^TA^TAx - b^TAx + \frac{1}{2}b^Tb \triangleq f(x)$$

A^TA always $H \succeq 0$

$\nabla f = A^TAx - A^Tb = \emptyset$ $\Rightarrow$ 2 cases to find the solution

(1) If A^TA is invertible $x^* = \underbrace{(A^TA)^{-1}A^T}_{\text{Left-pseudoinverse of } A}b$

(2) If A^TA is NOT invertible, but AA^T is invertible:

$$x^* = \underbrace{A^T(AA^T)^{-1}}_{\text{Right-pseudoinverse}}b \Leftarrow \begin{cases} Ax^* = AA^T(AA^T)^{-1}b \\ A^TAA^T(AA^T)^{-1}b - A^Tb = A^Tb = A^Tb - A^Tb = \emptyset \end{cases}$$

$$\hookrightarrow \begin{cases} f(x) = \emptyset \\ \nabla f = \emptyset \end{cases}$$

SUMMARY

OBJECTIVE: minimize $x^* = \arg\min_x \|Ax-b\|^2 \Leftrightarrow A^* = A^Tb$

SOLUTION: A^+ is computed using Singular Value Decomposition (SVD) (numerical method).

From Joint to cartesian space (task space)

We know that $\dot{x} = J\dot{q} + \dot{J}\dot{q}$ $\quad \ddot{x}^d = \ddot{x}^r + K_d(\dot{x}^r - \dot{x}) + K_p(x^r - x)$ [$\ddot{x}$ → acceleration end-effector]

We can compute $\ddot{q}^d$ as: $\begin{cases} \ddot{q}^d = \overbrace{J^+}^{\text{PSEUDOINVERSE}}(\ddot{x}^d - \dot{J}\dot{q}) \\ \tau^d = M\ddot{q}^d + h \end{cases}$ $\Rightarrow$ PID

If we treat the problem as an optimization problem:

$$(\tau^*, \ddot{q}^*) = \underset{\ddot{q},\tau}{\arg\min} \|\underbrace{J\ddot{q} + \dot{J}\dot{q}}_{\ddot{x}} - \ddot{x}^d\|^2 \equiv \|\ddot{x} - \ddot{x}^d\|^2 \quad \text{s.t. } M\ddot{q} + h = \tau$$

↳ It is still a least square program $\Rightarrow$ $\underbrace{J\ddot{q}}_{\text{Linear}} + \underbrace{\dot{J}\dot{q}}_{\text{const}} - \underbrace{\ddot{x}^d}_{\text{const}}$

We can add same constraints as seen before.

Task Models

- Generalize concept of e-e control
- Task = control objective
- Describe task with function $e(\cdot)$ (ERROR) to minimize
- Assume that $e(\cdot)$ describes error btw real and reference trajectory:

$$e(x,u,t) = \underbrace{y(x,u)}_{\text{Actual state of task}} - \underbrace{y^r(t)}_{\text{Desired trajectory}} \quad x \triangleq \underbrace{(q,\dot{q})}_{\text{Robot state}} \quad u \triangleq \tau$$

- We cannot directly minimize $e(\cdot)$ if it depends on q and $\dot{q}$, which are not decision variables of the LSP.

↳ IDEA: impose dynamics of e (e.g. $\dot{e}$ or $\ddot{e}$) such that: (I) $\lim_{t\to\infty} e(t) = 0$ (stability)

(II) Dyn. of e is affine function of $(\ddot{q},\tau)$ $(Ax+b)$

Il task è l'obiettivo che vogliamo raggiungere, e vogliamo farlo minimizzando l'errore

CASE #1 VELOCITY TASK FUNCTION

L'errore dipende dalla velocità dei giunti

$$e(x, u, t) = e(\dot{q}, t) = y(\dot{q}) - y^r(t)$$

[For instance: $y(\dot{q})$ = angular momentum]

$$\dot{e} = \dot{y} - \dot{y}^r = \frac{\partial y}{\partial \dot{q}} \frac{d\dot{q}}{dt} - \dot{y}^r = J\ddot{q} - \dot{y}^r$$

Choose $\dot{e}$ such that $e \to 0$. Let's take a linear feedback:

$\dot{e} = -Ke$* $K > 0 \Rightarrow \lim_{t \to \infty} e(t) = 0$

*Since $e = const$, $\forall K$ $-Ke = const$

$$J\ddot{q} - \dot{y}^r = -Ke$$

$$\underbrace{J}_{A}\ddot{q} = \underbrace{\dot{y}^r - Ke}_{b} \Rightarrow A\ddot{q} - b = 0 \Rightarrow \|A\ddot{q} - b\|^2$$

x risolvere usiamo i minimi quadrati e risolviamo per $\ddot{q}$

CASE #2 : CONFIGURATION TASK FUNCTION

L'errore deriva dalla posizione dei giunti

$e(x, u, t) = e(q, t) = y(q) - y^r(t)$

$\dot{e} = \dot{y} - \dot{y}^r = {}^{\partial y}/_{\partial q}\ {}^{dq}/_{dt} - \dot{y}^r = J\dot{q} - \dot{y}^r$ Not enough, I need it in function of $\ddot{q}$

$\ddot{e} = J\ddot{q} + \dot{J}\dot{q} - \ddot{y}^r$ Linear function of $\ddot{q}$

Let's impose linear dynamics: $\ddot{e} = -K_p e - K_d \dot{e}$ $K_p > 0$, $K_d > 0$

$J\ddot{q} + \dot{J}\dot{q} - \ddot{y}^r = -K_p e - K_d \dot{e}$

$$\underbrace{J}_{A}\ddot{q} = \underbrace{\ddot{y}^r - \dot{J}\dot{q} - K_p e - K_d \dot{e}}_{b} \Rightarrow \text{Minimize } \|A\ddot{q} - b\|^2$$

! The main difference between configuration and velocity task function is that in the fist there are 2 feedback gains (K_d, K_e), while in the second there is only 1.

CASE #3 : CONTROL-INPUT TASK FUNCTION

Control inputs $u \triangleq \tau$ are part of the decision variables.

Therefore $e(x, u, t) = e(u, t)$ must be affine: $e(u, t) = A_u(t)u - b(t)$

If it's not, I cannot use it in a Q.P.-based controller.

Example: $\|u\|^2$ or $\|u - u^r\|^2$

SUMMARY

TASK FUNCTIONS
- Affine function of $u \triangleq \tau$
- Nonlinear function of x:
  - for function of $\dot{q} \Rightarrow \dot{e}$
  - for function of $q \Rightarrow \ddot{e}$

In the end we have affine function of $(\ddot{q}, \tau)$:

$$g(z) = \underbrace{\begin{bmatrix} A_x & A_u \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \ddot{q} \\ \tau \end{bmatrix}}_{z} - b \Rightarrow \min \|g(z)\|^2$$

Task-Space Inverse Dynamic [TSID]

$\underset{z=(\ddot{q},\tau)}{\text{minimize}} \ \frac{1}{2}\|Az-b\|^2$ subject to $[M \mid -I]\, z = -h$ → Modo compatto per scrivere $M\ddot{q} - \tau = -h$

$z = \begin{bmatrix} \ddot{q} \\ \tau \end{bmatrix}$

For example:

- Joint-space control: $A = [I, 0]$, $b = \ddot{q}^d$ $\Rightarrow \underset{z=(\ddot{q},\tau)}{\min} \frac{1}{2}\|\ddot{q} - \ddot{q}^d\|^2$
  $Az - b = \ddot{q} - \ddot{q}^d$
- End-Effector control: $A = [J, 0]$, $b = \ddot{x}^d - \dot{J}\dot{q}$ $\Rightarrow \underset{z=(\ddot{q},\tau)}{\min} \frac{1}{2}\|J\ddot{q} + \dot{J}\dot{q} - \ddot{x}^d\|^2$
  $Az - b = J\ddot{q} + \dot{J}\dot{q} - \ddot{x}^d$ ($J\ddot{q} + \dot{J}\dot{q} - \ddot{x}^d = \ddot{x} - \ddot{x}^d$)
- Under-Actuated systems (Less actuators than DoFs):
  $\underset{z=(\ddot{q},\tau)}{\text{minimize}} \ \frac{1}{2}\|Az-b\|^2$ s.t. $[M \mid -S^T]\, z = h$ — vector that contain 0 in passive joints

TSID FOR RIGID CONTACTS (constraints for motion)

posizione del punto di contatto nello spazio

- $c(q) = \text{const}$ — contact points don't move
- $J\dot{q} = 0$ — contact point velocities are null
- $J\ddot{q} + \dot{J}\dot{q} = 0$ — contact point accelerations are null

J is the Jacobian of $c(q)$: $J \triangleq \partial c(q)/\partial q$

↳ se il robot tocca un oggetto, il punto di contatto non può muoversi

↳ vincolo + importante, poiché il nostro controller (QP) comanda $\ddot{q}$ e τ.

CONTACT FORCE F is a decision variable in QP:

$\underset{z=(\ddot{q},\tau,F)}{\min} \|Az-b\|^2$ s.t. $\begin{cases} M\ddot{q} + h = S^T\tau + J^T F \\ J\ddot{q} + \dot{J}\dot{q} = 0 \end{cases}$ (coppie ai motori)

Quando il robot entra in contatto, non solo deve eseguire il task, ma deve anche "obbedire" a nuovi vincoli imposti dal contatto con un oggetto

New constraint added

⇕

Single constraint in matrix form →

$$\begin{bmatrix} M & -J^T & -S^T \\ J & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ F \\ \tau \end{bmatrix} = \begin{bmatrix} -h \\ -\dot{J}\dot{q} \end{bmatrix}$$

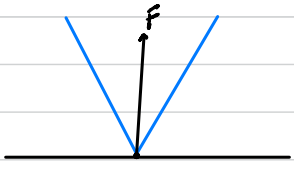
SOFT CONTACT ⇒ F = COST — F è sconosciuta, il sistema la calcola per mantenere il vincolo

HARD CONTACT ⇒ F depends on how much force I'm applying

↳ F fa parte di una variabile di costo. Ad esempio possiamo applicare una forza specifica minimizzando $\|F - F^d\|^2$

FRICTION CONE CONSTRAINTS

La forza deve essere fisicamente realistica

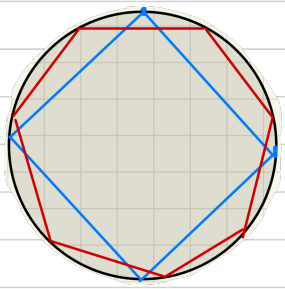


$\|F_T\|^2 \le \mu F_N^2$

F_T: tangential force (Attrito) — F_N: Normal force

μ: friction coefficient

→ Friction force must be stay in a "friction cone"



Approximate quadratic cone with linear pyramidal cone:

$BF \le 0$

Include this in TSID's QP.

Multy-Task Control

For complex (redundant) robots typically one has multiple control objectives, e.g. N task, each defined by a task function $g_i(z) \triangleq \|A_i z - b_i\|^2 \quad i = 1, \dots, N$

TSID QP: $\min_{z=(\ddot{q},\tau,F)} \sum_{i=1}^{N} w_i g_i(z) = \|Az - b\|^2$ s.t. $\begin{bmatrix} M & -J^T & -S^T \\ J & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ F \\ \tau \end{bmatrix} = \begin{bmatrix} \overset{\text{Dynamic}}{-h} \\ \underset{\text{Contact}}{-\dot{J}\dot{q}} \end{bmatrix}$ → HARD CONSTRAINTS (Physics Laws)

↳ Minimizzare la somma pesata di tutti i task

$w_i \in \mathbb{R}^+ \triangleq$ user-defined weight. Large $w_i \Rightarrow$ high importance to task i

Practical Implementation

Most industrial robots only allow for joint Pos/Vel commands. They have high joint friction (hard to model and to compensate) => hard to achieve torque control.

Joint friction is mainly a consequence of high gear ratios.

What can I do if I cannot send torque commands?

(1) Solve QP as usual to compute $z^* = (\ddot{q}^*, \tau^*)$

(2) Integrate $\ddot{q}^* \Rightarrow \dot{q}^d := \dot{q}^d + \Delta t \cdot \ddot{q}^*$

(3) Send joint velocity commands $\dot{q}^d$

2. OPTIMAL CONTROL

→ No need for reference trajectory
→ Objective clearly specified through cost function
→ Can handle constraints

O.C. METHOD FAMILIES

	OPTIMIZE ⇒ DISCRETIZE (Continuous Time)	DISCRETIZE ⇒ OPTIMIZE (Discrete Time)
Global	② Hamilton-Jacobi-Bellman (HJB)	① Dynamic Programming (DP)
Local	③ • Pontryagin Maximum Principle (PMP) • Calculus of variations • Indirect method	④ • Direct method

NOTE BOOK numerical_optim

Try to find best state $x(t)$ and control $u(t)$ trajectories in a time interval $t \in [0, T]$

O.C. PROBLEM

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}} \int_0^T e(x(t), u(t), t)\,dt + e_f(x(t))$$

(T: Fixed length of the Problem; e: RUNNING COST; e_f: TERMINAL COST (depends only on the terminal state); $x(\cdot), u(\cdot)$: Decision variables; whole expression: COST FUNCTION)

$$\text{subject To } \dot{x}(t) = f(x(t), u(t), t) + e_f(x(T)) \quad \forall t \in [0, T]$$

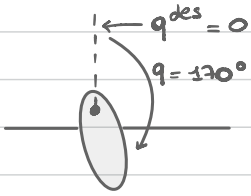
(Il robot deve seguire le leggi della fisica → Dynamics)

$$x(0) = x_0$$ Initial condition (Not mandatory)

$$g(x(t), u(t), t) \leq 0$$ Path constraints $\forall t \in [0, T]$

- $x(\cdot), u(\cdot)$ are trajectories ⇒ infinite-dimensional objects → stiamo cercando intere funzioni, non numeri
- constraints are infinitely many
- ≠ optimization problem (∞ time instant between $[0, T]$)

EXAMPLE Pendulum



$x = (q, \dot{q})$ state of the system
$u = \tau$ control input

$$\min_{x(\cdot),u(\cdot)} \int_0^T \left(w_q\, q(t)^2 + w_{\dot{q}}\, \dot{q}(t)^2 + w_\tau\, \tau(t)^2 \right) dt + 10^2\, q(t)^2$$

($w_q q(t)^2$: How much fast pendulum reach desired $q = 0$; $w_\tau \tau(t)^2$: Torque and acceleration are linear dependent so we can optimize one)

$$\text{subject To} \quad \underbrace{I\ddot{q} = \tau + mg\sin(q)}_{\text{System dynamic of a pendulum}} \Rightarrow \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ I^{-1}(\tau + mg\sin(q)) \end{bmatrix} \quad \forall t$$

$$|\tau(t)| \leq \tau^{max} \quad \forall t \in [0, T]$$

$$q^{min} \leq q(t) \leq q^{max} \quad \forall t \in [0, T]$$

$$q(T) = \emptyset$$

NOTE:

- $10^2 q(t)$ terminal cost must be in function only of the terminal state
- $q(t) = \emptyset$ model choice to set the final position to joint angle $q = 0$. If I don't specify a terminal constraint, I need to add a weight to the terminal cost. Same for joint velocities, we can define $\dot{q}(t) = 0$.
- $w_q q(t)^2$ It's possible to use the absolute value instead of square. With square you are more tolerant with small errors and penalize more big errors.

Non Linear Programming [2.2 Notebook]

Non linear progamming has as objective to minimize a non linear cost function $f(x)$, that have equality and disequality constraints.

For an **UNCONSTRAINED problem**, to verify that x^* is a LOCAL MINIMA, we have to check:

(1) $\nabla f(x^*) = 0$

(2) $\nabla^2 f(x^*) > 0$ Hessian matrix must be positive. Verify that x^* is a minimum.

↳ To find a solution we use iterative numerical methods:

- Gradient descent $x_{k+1} = x_k + \gamma \nabla f(x_k)$ converge slowly
- NEWTON METHOD $x_{k+1} = x_k - (\nabla^2 f)^{-1} \nabla f$ use 2° order informations, faster convergence.

  This method is unstable, so it's necessary two improvements
  - Hessian regularization: $H + \lambda I$ guarantee that direction is always descent
  - Line search: we looking for an optimal step length $a \in [0,1]$, to prevent divergences.

If we have a **CONSTRAINED problem** we have to find x^* that satisfy the constraints.
KKT conditions generalize gradient concept and represent necessary conditions for optimality.

- LAGRANGIAN FUNCTION: combines original cost function with constraints, weighted by coefficients called Lagrange multipliers (λ for eq., μ for dis.).

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$$

$g(x)$ = equality constraints; $h(x)$ = inequality constraints

- KKT CONDITIONS: For a point x^*, to be a minimum, must be exist a set of λ, μ such that:

1) Stationarity: $\nabla_x \mathcal{L}(x, \lambda, \mu) = 0$ → we can't move in any direction without violate any of the constraints.

2) Primal Admissibility: x^* must respect all constraints $\begin{cases} g(x^*) = 0 & \text{equality constraints} \\ h(x^*) \leq 0 & \text{inequality constraints} \end{cases}$

3) Dual admissibility: $\mu \geq 0$ this the "cost" of the constraint. If $\mu = 0$ that constraint doesn't limit us.

4) Complementary Slackness: $\mu_i h_i(x^*) = 0 \ \forall i$ → x^* can be only on the frontier of the constraint or inside it. If x^* is inside of admissible region of the constraint, it is not affected by it.

EQUALITY CONSTRAINTS $g(x^*) = 0$

Define feasible cone: $F(x) = \{ d \mid \nabla g(x)^T d = 0 \}$ Insieme di tutte le direzioni d in cui posso muovermi partendo da d. $\nabla g(x)^T d = 0$ → d deve essere perpendicolare al gradiente del vincolo $\nabla g(x)$

Set of descent directions: $\nabla(x) = \{ d \mid d^T \nabla f(x) < 0 \}$ Insieme delle direzioni che riducono $f(x)$

Local minimum $x \Rightarrow \nabla(x) \wedge F(x) = 0$ In un minimo non esiste nessuna direzione che sia contemporaneamente ammissibile e di discesa. Se così non fosse, d mi porterebbe in una direzione migliore, perciò x^* non sarebbe un minimo

KKT CONDITIONS $\begin{cases} g(x) = 0 \\ d^T \nabla f(x) \geq 0 \ \forall d \in F(x) \Leftrightarrow d \in \text{null}(\nabla g(x)^T) \end{cases}$ (se non ci sono direzioni ammissibili, la funzione sale o rimane piatta) $\Rightarrow \nabla f(x) = \nabla g(x) \lambda$

↳ I 2 gradienti sono paralleli

Define Lagrangian $\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$

↓

KKT conditions: $\nabla \mathcal{L}(x, \lambda) = 0 \Leftrightarrow \begin{cases} \nabla f(x) + \nabla g(x) \lambda = 0 & \text{Derivation wrt } x \\ g(x) = 0 & \text{Derivation wrt } \lambda \end{cases}$

INEQUALITY CONSTRAINTS

Define set of ACTIVE constraints $\mathcal{A}(x) = \{ i \mid h_i(x) = 0 \}$

↳ Active means that we are on boundaries, so that constraint limits us. Otherwise the constraint is inactive and we can move in any direction inside the region.

Feasible cone $F(x) = \{ d \mid \nabla g(x)^T d = 0 \;,\; \nabla h_i(x)^T d \leq 0 \;,\; i \in \mathcal{A}(x) \}$ — Considering only active constraints, by moving in any direction d they can only decrease or remain the same

KKT conditions $\begin{cases} g(x) = 0, \quad h(x) \leq 0 \\ d^T \nabla f(x) \geq 0 \quad \forall d \in F(x) \end{cases} \iff \nabla f(x) + \nabla g(x) \lambda + \nabla h_{\mathcal{A}}(x) \mu_{\mathcal{A}} = 0$

$\mu_{\mathcal{A}} \geq 0$

$$\iff \begin{cases} \nabla f(x) + \nabla g(x) \lambda + \nabla h(x) \mu = 0 \\ \mu \geq 0 \\ \mu h(x) = 0 \end{cases}$$

complementary condition $\Rightarrow$ if $h(x) = 0 \rightarrow \mu_i$ can be positive

if $h(x) < 0 \rightarrow \mu_i = 0$

SOLVING KKT CONDITIONS

$\min f(x)$

s.t. $g(x) = 0$

$h(x) \leq 0$

$\Rightarrow$ KKT system

$$\begin{cases} \nabla_x \mathcal{L}(x, \lambda, \mu) = 0 \\ g(x) = 0 \\ h(x) \leq 0 \\ \mu \geq 0 \\ \mu^T h(x) = 0 \end{cases}$$

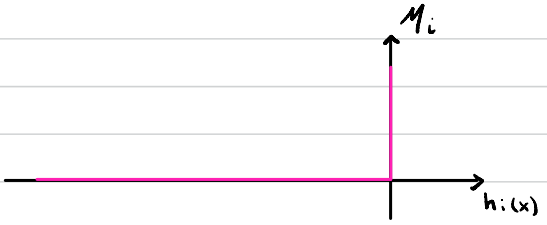
for equality constraints

extra for inequality constraints

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$$

Direct Methods

Newton method cannot be directly used to solve inequality constraints because solution space is not smooth.



KEY IDEA: discretize OCP $\Rightarrow$ use NLP solver to find local optimum

DISCRETIZATION: 1) parametrize state/control trajectories

2) enforce constraints on a time grid $t_0 < t_1 < \dots < t_N$

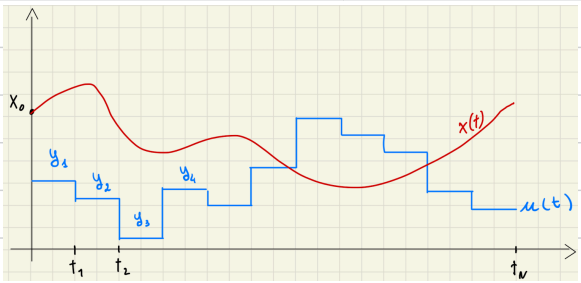
(1) SINGLE SHOOTING (sequential):

- discretize only $u(t)$
- compute $x(t)$ integrating dynamics
- No need to have dynamic constraints

} Discretizzo solo il controllo $u(t)$ e ne derivo $x(t)$ di conseguenza

2. Discretize $u(t) = y_i \quad \forall t \in [t_i, t_{i+1}]$

↓ optimization variable



b. Compute $x(t)$ from $u(t)$ and x_0 by integrating dynamics:

$$\begin{cases} \dot{x}(t) = f(x, u, t) \\ x(0) = x_0 \text{ fixed and known} \end{cases}$$

c. NLP

$$\min_{y} \int_0^{T_f} \underbrace{l(x(t,y), u(t,y), t)\,dt}_{\text{Running cost}} + \underbrace{l_f(x(t_N, y))}_{\text{terminal cost}}$$

La funzione di costo viene calcolata integrando i costi lungo la traiettoria $x(t)$ ottenuta

$$\text{s.t. } g(x(t_i, y), u(t_i, y), t_i) \leq 0 \quad i = 0, \dots, N$$

I vincoli vengono imposti solo nei punti della griglia temporale

Discretize path constraints

Running cost integral can be computed when integrating dynamics:

$$c(t) = \int_0^t l(\cdot,\cdot)\,dt$$

$$\begin{cases} \dot{c}(t) = l(\cdot,\cdot) \\ c(0) = 0 \end{cases} \Rightarrow \quad \bar{x} = (x, c), \quad \dot{\bar{x}} = \begin{bmatrix} f(x, u, t) \\ l(x, u) \end{bmatrix}$$

NUMERICAL INTEGRATION

given ODE: $\dot{x}(t) = f(x(t), t)$, $x(0) = x_0$

compute $x(t)$ $\forall t \in [0, T]$

If first derivatives of f are bounded ⇒ UNIQUE SOLUTION

EXPLICIT EULER

$$\dot{x} = \lim_{h \to 0} \frac{x(t+h) - x(t)}{h}$$

$$hf(x,t) \simeq x(t+h) - x(t) \Rightarrow x(t+h) = x(t) + hf(x,t)$$

small h ⇒ ↑ acc. ↑ comp. cost

RUNGE-KUTTA METHODS

$x_{n+1} = x_n + h \sum_i^q b_i K_i$ approximate average $\dot{x}$ with weighted average of K_i, with weights b_i, with $\sum b_i = 1$, $b_i \geq 0$

$K_i = f\left(x_n + h \sum_j^q a_{ij} K_j,\ t_n + c_i h\right)$ approximate value of $\dot{x}$ at time $t_n + c_i h$, with $c_i \in [0, 1]$

- a_{ij} and c_i say when and where evaluate dynamics
- q is the order of the method → how many evaluations of dynamics to compute.
- for $q > 1$ exists many versions of same order P (consistency).

SUMMARY

- Remove x from decision variables
- Discretize $u(t) = y_i \ \forall t \in [t_i, t_{i+1}]$
- Compute x by integrating dynamics $\dot{x} = f(x, u, t)$ (I recover x integrating)
- Remove dynamics from constraints

- High-order integration schemes (typically) more efficient
- Differentiation of integration scheme often computed using automatic diffentiation libraries.
- Implicit schemes typically needed for stiff dynamics

2 COLLOCATION

Discretizzo sia $x(t)$ (s_i) che $u(t)$ (y_i)

a. Discretize $x(t)$ with polynomials (e.g. order $K=0$) of fine grid:

$x(t) = s_i \quad \forall t \in [t_i; t_{i+1}]$ → one polynomial for each time interval

Sia lo stato che il controllo vengono parametrizzati. Vengono rappresentati da polinomi a tratti, le variabili di ottimizzazione diventano i coefficienti di questi polinomi (o i loro valori s_i, y_i nei punti di griglia)

b. Replace Dynamics $\dot{x} = f(x,u)$ with: approssimazione

$$\underbrace{\frac{s_{i+1} - s_i}{t_{i+1} - t_i}}_{\text{Approximation of dynamics}} - \underbrace{f(s_i, y_i)}_{\text{Dynamics}} = 0 \quad \text{rewriting} \rightarrow \quad c_i(s_i, s_{i+1}, y_i) = 0$$

↳ la dinamica diventa un VINCOLO

Approximation of dynamics has to be equal to real dynamics to have best approximation possible

c. Approximate cost integral: $\int_{t_i}^{t_{i+1}} \ell(x(t), u(t))\,dt \simeq \ell(s_i, y_i)(t_{i+1} - t_i) \triangleq l_i(s_i, y_i)$

d. NLP (sparse)

$$\min_{s,y} \sum_0^{N-1} l_i(s_i, y_i) + l_F(s_N)$$ Minimize sum of approximated costs at each interval + terminal cost

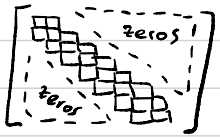
$$\text{s.t.} \begin{cases} s_0 - x_0 = 0 \quad \text{(initial conditions)} \\ c_i(s_i, s_{i+1}, y_i) = 0 \quad i = 0, \dots, N-1 \leftarrow \text{discretized dynamics} \\ g_i(s_i, y_i) \le 0 \quad i = 0, \dots, N-1 \leftarrow \text{discretized path constraints} \end{cases}$$

La sequenza dei punti deve partire dalla condizione iniziale

Check constraints only at discrete time ist.

KKT matrix is SPARSE → most of the entry of the matrix are zero. This is useful because solvers can check sparsity and the precence of zeros lead to a more efficient computation.

$H_{ij} = \frac{\partial}{\partial s_i}\left(\frac{\partial}{\partial s_j} L\right) = 0 \quad \forall i \neq \{j, j+1, j-1\}$ $\quad H =$



3 MULTIPLE SHOOTING → hybrid between single shooting and collocation. Tries to solve instability problems in single shooting.

a. Discretize control $u(t)$ on coarse grid $t_0 < t_1 < \dots < t_N$ $\quad u(t) = y_i \ \forall t \in [t_i, t_{i+1}]$

b. Integrate numerically ODE in each interval $[t_i, t_{i+1}]$ state variable s_i:

$\dot{x}(t) = f(x_i(t), y_i) \quad t \in [t_i, t_{i+1}]$

$x_i(t_i) = s_i$ Instead of one single integration, compute N integration starting from s_i (start of interval)

Obtain $x_i(t)$, then numerically compute integrals: $l_i(s_i, y_i) = \int_{t_i}^{t_{i+1}} \ell(x_i(t), y_i)\,dt$

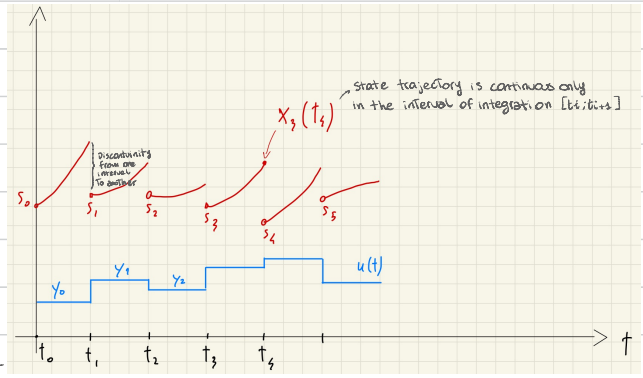
c. NLP

$$\min_{(s,y)} \sum_0^{N-1} l_i(s_i, y_i) + l_F(x_N)$$

$$\text{s.t.} \begin{cases} s_0 - x_0 = 0 \quad \text{initial conditions} \\ s_{i+1} - x_i(t_{i+1}, s_i, y_i) = 0 \quad \text{continuity constraints} \\ g_i(s_i, y_i) \le 0 \quad \text{path constraints} \end{cases}$$

Resulting of numerical integration from s_i

Starting point of next segment must be exactly the point in which is finished previous segment



DIRECT METHODS - SUMMARY

① SINGLE SHOOTING

- ⊕ Small problem
- ⊕ Need initial guess only for $u(t)$
- ⊕ Can use off-the-shelf ODE solvers
- ⊖ Cannot exploit knowledge of $x(t)$ in initialization
- ⊖ Numerical issues for unstable systems

③ MULTIPLE SHOOTING

- • Medium problem
- ⊕ Sparser than ①
- ⊖ Less sparse than ②
- ⊕ Unstable OK
- ⊕ Initialize $x(t)$
- ⊕ Can adapt time grid

② COLLOCATION

- ⊖ Large problem
- ⊖ Cannot adapt time grid
- ⊕ Sparse problem
- ⊕ Work well with unstable systems
- ⊕ Can initialize $x(t)$

Model Predictive Control (MPC)

Per controllare un sistema in TEMPO REALE risolvendo i problemi di controllo ottimo

- Use optimal control for controlling system (robot).

Non calcola la traiettoria di controllo una sola volta, ma riprende un processo in 3 fasi a ogni passo temporale

1. Solve a finite-horizon OCP using current state as initial state.

$$X^*, U^* = \underset{X,U}{\arg\min} \sum_{0}^{N-1} l(x_i, u_i) \quad \text{(collocation)} \quad \rightarrow \text{cost function is the sum of "transition cost" along horizon } N$$

$$\text{s.t.} \quad x_{i+1} = f(x_i, u_i) \quad i = 0, \dots, N-1 \quad \rightarrow \text{Finite-horizon}$$

$$x_{i+1} \in \mathcal{X} \quad u_i \in \mathcal{U} \quad \text{(admissible states and controls)} \qquad i = 0, \dots, N-1$$

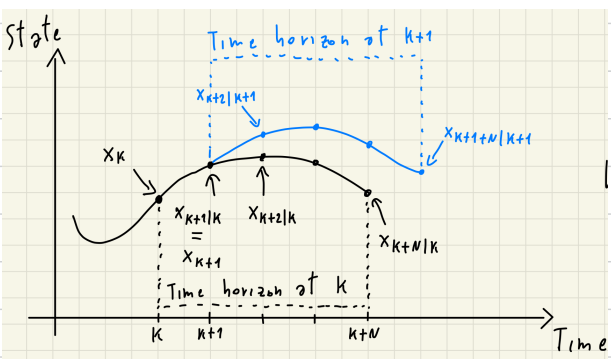
$$x_0 = x^{\text{mean}}$$

2. Apply computed control only for this step: $\tau = u_0^*$ (dell'intera sequenza di controllo ottimale calcolata, si applica solo il primo comando al sistema)
3. Repeat the process for the next step. In this way the control remain optimal despite changes in environment. For example, the precence of an obstacle can change set $\mathcal{X}$.

Al passo temporale successivo, il sistema si troverà in un nuovo stato (dovuto al comando u_0^* ed eventuali disturbi). L'intero processo ricomincia dal punto 1.

↳ **RECEDING HORIZON** ORIZZONTE MOBILE

OCP at time k and $k+1$ optimize over different horizons so can result in different Trajectories.



Even without disturbances, predicted and actual trajectory can be different. This happens because model $f(x,u)$ is just an approximation. MPC try to fix iteratively this error.

CHALLENGES
- FEASIBILITY: how to ensure feasibility and deal other cases? ASSICURARE CHE CI SIA SEMPRE UNA SOLUZIONE
- STABILITY: can MLP stabilize the system? MPC è in grado di far convergere il sistema senza oscillazioni incontrollate
- COMPUTATION TIME: is it sufficiently fast? Deve essere risolto tra un passo temporale e l'altro

INFINITE HORIZON MCP $N = \infty$ ⇒ Fornisce "garanzie teoriche", nella pratica si usa l'orizzonte finito

In this case pred.-act. trajectories are the same. This follows from Bellman's optimality principle:

Given optimal traj. $x^*(0), \dots, x^*(N)$, the subtraj. is optimal for OCP in horizon $[k, N]$ starting from $x(k)$. ⇒ optimal problem doesn't change at each step

With $N = \infty$, if cost $\ell(x,u) \geq \alpha \|x\| \ \forall x, u$ for some $\alpha > 0$, then having a finite cost implies stability.

Feasibility of the first OCP implies feasibility of all the OCP's.

IDEA: add terminal cost and constraints to finite-horizon OCP to mimic a infinite horizon.

$$V_N^*(\bar{x}) = \min_{U} \sum_{k=0}^{N-1} \ell(x_n, u_n) + \ell_f(x_n)$$

Estimation of total cost that would accumulate from N to ∞ — $\ell_f(x_n)$: terminal cost

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k) \quad k = 0, \dots, N-1$$
$$x_{k+1} \in \mathcal{X}, \ u_n \in \mathcal{U} \quad k = 0, \dots, N-1$$
$$x_0 = \bar{x}$$
$$x_N \in \mathcal{X}_f \quad \text{terminal constraint}$$

$x_N \in \mathcal{X}_f$ terminal constraint: force terminal state to finish in a terminal region $\mathcal{X}_f$

How to choose $\ell_f(\cdot)$, $\mathcal{X}_N$?

1 FEASIBILITY

Is the OCP going to be feasible at all sampling instants t?

- INPUT CONSTRAINTS ONLY → always feasible
- HARD STATE CONSTRAINTS → if $N < \infty$ there is no guarantee that OCP remains feasible, even in nominal case

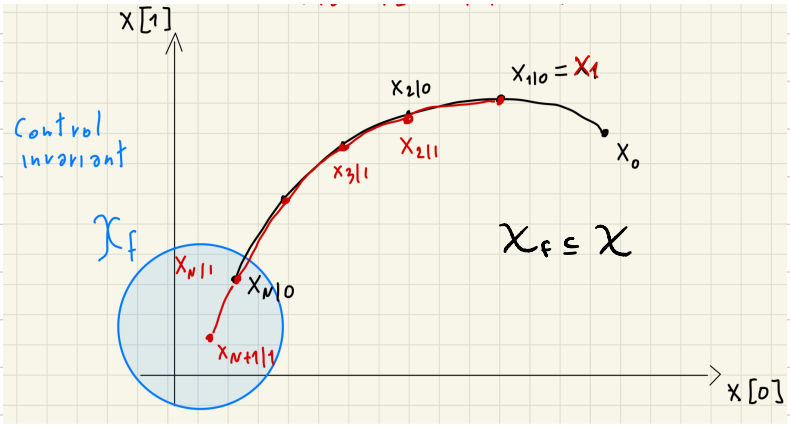
MAXIMUM OUTPUT ADMISSIBLE SET THEORY:
- $N < \infty$ is enough
- hard to know which value of N is enough
- computation time may be too high to that N

THEOREM If $\mathcal{X}_f$ is control invariant ⇒ MCP is recursively feasible

DEFINITION A set S is control invariant if $\forall x \in S, \exists u \in \mathcal{U}$ s.t. $f(x,u) \in S$

Per ogni stato che si trova in S, esiste un controllo u tale che lo stato successivo $f(x,u)$ rimane dentro S

VISUAL PROOF



$\mathcal{X}_f$ IS CONTROL INVARIANT SET: a system is of CI if, once you're in it, exists always a control that allows to remain in the set.

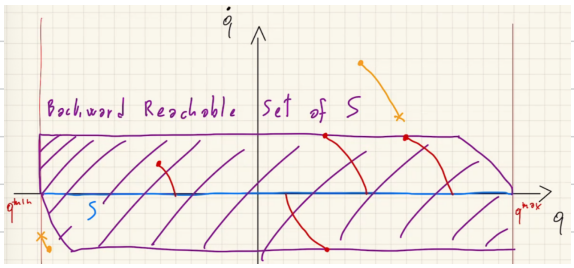
SUMMARY: an invariant set is like a safe region in which we can always find a solution

CONTROL - INVARIANT SETS

Set of equilibria $\mathcal{S}$ is always C.I., but small: $\mathcal{S} = \{x \in \mathcal{X} \mid \exists u \in \mathcal{U} : x = F(x,u)\}$

Can use backward reachable set of $\mathcal{S}$, i.e. states starting from which you can reach $\mathcal{S}$.

↳ Visto che $\mathcal{S}$ è troppo piccolo da raggiungere per MPC, cerchiamo di farlo arrivare in una regione da cui è possibile raggiungere $\mathcal{S}$



We can easily check whether a state $\bar{x} \in B_N(S)$ (N-step backward reachable set of S) by solving this OCP:

$$\min_{x,u} \quad 1$$

$$\text{s.t.} \quad x_{i+1} = f(x_i, u_i) \qquad i = 0, \dots, N-1$$
$$x_{i+1} \in \mathcal{X}, \; u_i \in \mathcal{U} \qquad i = 0, \dots, N-1$$
$$x_N = x_{N+1}$$
$$x_0 = \bar{x}$$

$$\begin{cases} \bar{x} \in B_N(S) & \text{if } \exists \text{ solution} \\ \bar{x} \notin B_N(S) & \text{otherwise} \end{cases}$$

- Use FUNCTION APPROXIMATION to learn an approximate representation of $B_N(S)$ from examples.
- To BUILD A DATASET, repeat the following steps:
  - sample state $\bar{x}$
  - if $\bar{x} \notin \mathcal{X}$ ⇒ add data point $(\bar{x}, 0)$ to dataset
    else try to solve OCP
  - if solution found ⇒ add $(\bar{x}, 1)$ to dataset
    else add $(\bar{x}, 0)$ to dataset
- train classifier to predict label 0/1 given state $\bar{x}$.
- use the classifier as terminal constraint in MPC.

(2) STABILITY

LYAPUNOV FUNCTION: $V: \mathbb{R}^n \to \mathbb{R}$ is exponential Lyapunov function if $\exists\, a_1, a_2, a_3 > 0$ s.t. $\forall x \in \mathcal{X}$:

↳ "Misura di energia" $V(x)$ del sistema. GOAL → stabilizzare all'origine $x=0$, il punto con energia minima

$$a_1|x| \le V(x) \le a_2|x| \implies V(x) > 0 \quad \forall x \neq 0$$

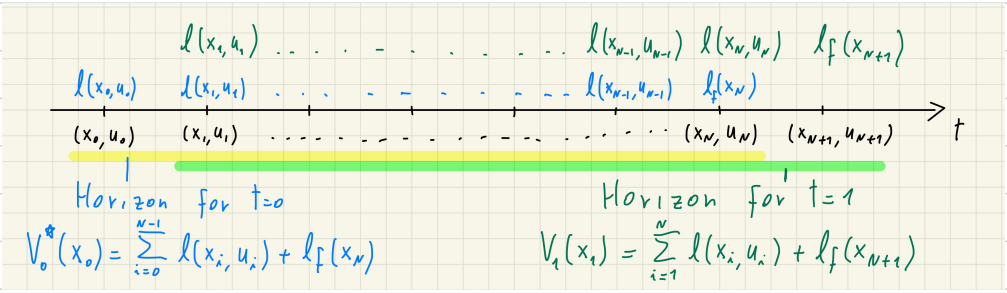
↳ $V(x)$ è una funzione convessa ≥ 0 ⇒ $V(0) = 0$

$$V(f(x)) - V(x) \le -a_3|x|$$

Ad ogni passo del sistema l'energia diminuisce

Th: IF $\exists$ Lyap. function ⇒ origin is exponentially stable in $\mathcal{X}$.

$$\|x_k\| \le c\,\gamma^k \|x_0\| \quad \text{for some } c > 0,\ \gamma \in [0,1]$$



Value function as a Lyapunov function

Is $V^*(x)$ a Lyapunov function?

$$V(x_1) - V^*(x_0) = \underbrace{\ell(x_N,u_N) + \ell_f(x_{N+1})}_{\leq 0 \text{ ASS. \#2}} - \ell_f(x_N) \underbrace{- \ell(x_0,u_0)}_{\leq -a_1|x_0| \text{ ASS. \#3}} \leq a_3|x_0| \quad \forall x_N \in \mathcal{X}_f$$

! Sufficient but not necessary because $V_1(x_1)$ is not optimal.

STABILITY ASSUMPTIONS

ASS #1: $f(0,0)=0$, $\ell(0,0)=0$, $\ell_f(0)=0$

ASS #2: $\forall x \in \mathcal{X}_f$, $\exists u \in \mathcal{U}$ s.t. $\ell_f(f(x,u)) - \ell_f(x) \leq -\ell(x,u)$ terminal cost decrease

ASS. #3: $\exists a_1 > 0$ s.t. $\forall x \in \mathcal{X}_N$, $\forall u \in \mathcal{U}$ $\mathcal{X}_N$ is set of states from which ocp has solution

$\ell(x,u) \geq a_1|x|$ cost lower bound

If #1, #2, #3 hold $\Rightarrow$ $V^*(\cdot)$ is an exponential Lyapunov function

Note that assuming $f(0,0)=0$ is not restrictive, if we want to stabilize $x^* \neq 0$, s.t. $f(x^*,u^*)=x^*$, i.e. x^* must be an equilibrium $\Rightarrow$ Define new state $\bar{x} = x - x^*$, control $\bar{u} = u - u^*$

dynamics:

$$\left.\begin{aligned} x^+ &= f(x,u) \\ x^* &= f(x^*,u^*) \end{aligned}\right\} \quad x^+ - x^* = f(\bar{x}+x^*, \bar{u}+u^*) - x^* \qquad \bar{x}^+ = \bar{f}(\bar{x},\bar{u})$$

So that $\bar{f}(0,0) = f(0+x^*, 0+u^*) - x^* = 0$

STABILITY OPTIONS:

(1) Pick $\mathcal{X}_f$ and ℓ_f such that: - $\mathcal{X}_f$ is control invariant

- $\forall x \in \mathcal{X}_f$ $\exists u \in \mathcal{U}$ s.t. $\ell_f(f(x,u)) - \ell_f(x) \leq -\ell(x,u)$

(2) Pick $\mathcal{X}_f = \{0\}$ $\Rightarrow$ Small basin of attraction (this is a special case of (1))

(3) Pick N "large enough" and set $\ell_f(x)=0$, $\mathcal{X}_f = \mathcal{X}$

↳ most common, especially for non linear systems. But large $N \Rightarrow$ large computation time

STABILITY EXTENSIONS

- Trajectory tracking can be cast as regulation of time-varying systems.
- Stability + recursive feasibility can still be ensured with time varying $\mathcal{X}_f$, $\ell_f(\cdot)$.
- If cost measures some kind of energy consumption $\Rightarrow$ does not satisfy $\ell(x,u) \geq a|x|$

$\Rightarrow$ Economic MPC

(3) COMPUTATION TIME

KEY IDEA: warm start, for example using the solution of previous OCP as initial guess for current OCP.

- Shift trajectory back by 1 step: $u_n^{guess} \leftarrow u_{k+1}^*$
- Use zero as initial guess for last time step $u_{N-1}^{guess} \leftarrow 0$
- Don't iterate until convergence, just do 1 Newton step. Possibly skip line search.
- Use hardware accelerators (e.g. GPU).

3. REINFORCEMENT LEARNING

- Tries to find global optimal policy
- Assumes Dynamic is unknown
- Finite size state/control spaces
- Assumes Dynamic is stochastic (consider friction change during motion)
- Infinite Horizon

} Differences with optimal control

Markov Decision Processes

Describe environment for RL problems.

Markov Property: FUTURE IS INDIPENDENT FROM THE PAST GIVEN THE PRESENT. $\mathbb{P}(x_{t+1}|x_t) = \mathbb{P}(x_{t+1}|x_1,\dots,x_t)$

State trausition probability $\mathcal{P}_{xx'} = \mathbb{P}(x_{t+1}=x' \mid x_t = x) \Longrightarrow$ matrix $\mathcal{P} = \begin{bmatrix} p_{11} & \dots & p_{1n} \\ \vdots & & \vdots \\ p_{n1} & \dots & p_{nn} \end{bmatrix}$ Each row sum = 1 ↳ due To Total probability law

Markov Process is defined as the tuple $<\mathcal{X}, \mathcal{P}>$ where $\mathcal{X}$ finite set of States

↳ Markov Chain

Since $\mathcal{P}$ has all nonnegative entries, and it's irreducible, by Perron-Frobenius theorem we know that:

PROP 1. The largest eigenvalue r of $\mathcal{P}$ satisfies: $\min_i \sum_j p_{ij} \le r \le \max_i \sum_j p_{ij} \Rightarrow r=1$

PROP 2. All eigenvalues of $\mathcal{P}$ are smaller than 1: $\lambda_i(\mathcal{P}) < 1 \ \forall i$

↳ !! IMPORTANT PROPERTIES

In MP dynamics is encoded as pdf: $\mathbb{P}(x^+ \mid x)$ Probability of transitioning from state x to next state x^+.

In OCP, for det. systems, dynamics was encoded as $x^+ = f(x)$ or $x^+ = f(x, \underbrace{w}_{\text{uncertainty}})$ $\Longrightarrow$ TRY TO USE THIS ONE

REWARD PROCESS

Tuple $<\mathcal{X}, \mathcal{P}, C, \gamma>$ → C = cost function $C_x = \mathbb{E}[\ell_{t+1} \mid x_t =]$ we work with expected values $\Rightarrow$ stochastic

↳ $\gamma \in [0,1]$ = discount factor, defined by user. It prevents cost To go to infinity

Negative cost $\Rightarrow$ reward

COST TO GO

Total discounted cost from time t to ∞: $J_t = \ell_t + \gamma \ell_{t+1} + \dots = \sum_{k=0}^{\infty} \gamma^k \ell_{t+k}$ somma di tutti i costi futuri che incontrerai dal tempo t a ∞

↳ γ represents preference for later costs over immediate cost

- $\gamma \to 0$ myopic evaluation
- $\gamma \to 1$ far sighted evaluation

} Found a trade-off to prevent $J_t = \infty$ (since we sum till infinity)

VALUE FUNCTION

Cost starting from state x: $V(x) = J_t(x_t = x) = \ell_t + \gamma(\ell_{t+1} + \gamma \ell_{t+2} + \dots)$

↳ can be decomposed in: $V(x) = \ell(x) + \gamma V(f(x))$ $\Rightarrow$ Bellman Equation — IL valore di stare in uno stato è il costo che pago adesso + il valore scontato di dove finisco dopo

↳ matrix form $V = C + \gamma P V \Rightarrow \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} \ell(1) \\ \vdots \\ \ell(n) \end{bmatrix} + \gamma \begin{bmatrix} p_{11} \dots p_{1n} \\ \vdots \\ p_{n1} \dots p_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$ → Expected value to go from state 1 to other n states

Linear equation in V: $V = (I - \gamma P)^{-1} C$ the solution is found with iterative methods

Valore esatto di ogni stato

Valore atteso futuro

HDP

Tuple $\langle X, U, P, C, \gamma \rangle$ → Introduced control input u (action a in RL terminology)

$P_{xx'}^{u} = \mathbb{P}(X_{t+1} = x' \mid X_t = x, u_t = u)$
$C_x^u = \ell_t(X_t = x, u_t = u)$
} cost and prob. depend also on control input

Policy ⇒ acts like a controller, its a distribution over control inputs given states: $\pi(u \mid x) = \mathbb{P}(u_t = u \mid X_t = x)$

We will assume deterministic policies $u = \pi(x)$. We can use stochastic policie during policy search process

ACTION-VALUE FUNCTION

It's the cost starting from state x, applying input u, and then following policy π. You don't know the future cost, it depends on the next action you decide to take.

$$Q^{\pi}(x,u) = J_t(X_t = x, u_t = u, u_{k>t} \sim \pi)$$

Can be decomposed as: $Q^{\pi}(x,u) = \ell(x,u) + \gamma Q^{\pi}(f(x,u), \pi(f(x,u))) = \ell(x,u) + \gamma V^{\pi}(f(x,u))$

costo immediato + il valore scontato dello stato in cui "atterri"

where: $V^{\pi}(x) = Q^{\pi}(x, \pi(x))$ Relation between value function and action-value function

first action it's the only that doesn't came from action set π

OPTIMAL VALUE FUNCTION

- Minimum value function over all policies: $V^*(x) = \min_{\pi} V^{\pi}(x)$
- Optimal action-value function: $Q^*(x,u) = \min_{\pi} Q^{\pi}(x,u)$
- Optimal policy: $\pi^* \leq \pi \ \forall \pi$ [where $\pi \leq \pi'$ if $V^{\pi}(x) \leq V^{\pi'}(x) \ \forall x$]

OPTIMAL POLICY

Minimize Q^* over u: $\pi^*(x) = \arg\min_{z \in U} Q^*(x,z)$

Se conosci la funzione Q ottimale, non devi pianificare nulla. Ti basta essere avido (greedy) e scegliere l'azione che minimizza il costo istantaneo combinato col futuro. Ecco perché in Reinforcement Learning (es. Q-Learning) cerchiamo disperatamente di stimare Q: se hai Q, hai il controllo perfetto.

↳ If we know $Q^*(x,u)$ we immediatly have optimal policy $\pi^*(x) = \min_{u} \ell(x,u) + \gamma V(f(x,u))$

BELLMAN OPTIMAL EQUATION

$$V^*(x) = \min_{u} Q^*(x,u)$$

$$Q^*(x,u) = \ell(x,u) + \gamma V^*(f(x,u))$$

$$V^*(x) = \min_{u} \ell(x,u) + \gamma V^*(f(x,u))$$

$$Q^*(x,u) = \ell(x,u) + \gamma \min_{u'} Q^*(f(x,u), u')$$

- Non linear
- No closed-form solution
- Iterative Algorithms

LINEAR: $V^* = \ell + \gamma P V^*$ | NON LINEAR: $V^* = \min \ell + \gamma P V^*$

→ Non ancora RL, fa parte di optimal control methods

Dynamic Programming

Assume full knowledge of MDP, we have 2 classes of problems:

1. PREDICTION
   - Input: MDP $\langle X, U, P, C, \gamma \rangle$ Policy π
   - Output: Value function V^π
2. CONTROL
   - Input: MDP
   - Output: Optimal Value Function V^*, Optimal Policy π^*

Prediction (Evaluation): Ti do una policy fissa π. Tu dimmi quanto vale (Vπ). È un lavoro da contabile: non cambi nulla, calcoli solo il disastro (o il successo) futuro.

Control (Optimization): Non mi interessa la policy attuale. Voglio la migliore policy possibile (π*) e il miglior valore possibile (V*)

FINITE HORIZON	INFINITE HORIZON
- theory is simpler	- complex theory
- used in optimal control	- used in RL
- Policy and value depends on time	- Policy and value are stationary
	- good approximation of problems with finite but long horizon

All algorithm of DP rely on **Bellman operators**.

- BELLMAN EXPECTATION OPERATOR $T^\pi(V) = C^\pi + \gamma P^\pi V$ — It's following a Markov Chain — Modifies a given value function
  Aggiorna il valore basandoti sulla media pesata delle probabilità, seguendo la policy attuale.
- BELLMAN OPTIMALITY OPERATOR $T(V) = \min_{u \in U} C^u + \gamma P^u V$
  Aggiorna il valore assumendo che d'ora in poi farò sempre la mossa migliore

$$(TV)(x) = \min_{u \in U} \ell(x,u) + \gamma V(f(x,u)) \leftarrow \text{Different Notation}$$

(Dynamics Notation)

PREDICTION PROBLEM

PROBLEM: evaluating a given policy π

SOLUTION: Start from arbitrary estimate of value function V_0

Apply iteratively Bellman expectation operator: $V_{k+1}(x) = \ell(x, \pi(x)) + \gamma V_k(f(x, \pi(x)))$ $\forall x \in X$

↳ $V_{k+1}(x) = T^\pi(V_k)$

Guaranteed to converge to V^π: $\lim_{k \to \infty} V_k = V^\pi$

Stop when $\|V_{k+1} - V_k\| <$ treshold where treshold $\to 0$

CONTROL: POLICY ITERATION

PROBLEM: find optimal policy π

SOLUTION: start with arbitrary policy π_0

For $k = 0 \dots \infty$

- Evaluate $\pi_k \Rightarrow V^{\pi_k}$
- Improve policy acting greedily wrt V^{π_k} → Una volta che hai i valori veri, aggiorna la policy scegliendo l'azione che minimizza il costo immediato + costo futuro

$$\Rightarrow \pi_{k+1}(x) = \arg\min_u \ell(x,u) + \gamma V^{\pi_k}(f(x,u))$$

Always converges to π^*: $\lim_{k \to \infty} \pi_k = \pi^*$

Evaluate exactly a policy can take many iterations. If we compute an approximation of V^{π_k} we use m_k evaluation iteration to compute V^{π_k}.

Under some mild assumptions it converges to V^*.
- for $m_k = \infty$ we recover policy iteration
- for $m_k = 0$ we get value iteration

Every time you update V you use a policy that is greedy wrt V:

$$V^k(x) = \ell(x, \pi^k) + \gamma V^{k-1}(f(x, \pi^*)) \quad \text{with } \pi^*(x) = \arg\min_u \ell(x,u) + \gamma V^{k-1}(f(x,u))$$

$$\Rightarrow V^k(x) = \min_u \ell(x,u) + \gamma V^{k-1}(f(x,u))$$

CONTROL: **VALUE ITERATION**

Algorithm:

- Start with arbitrary value function V_0

For $k=0 \dots \infty$

For any $x \in \mathcal{X}$

$$V_{k+1}(x) = \min_{u \in \mathcal{U}} \ell(x,u) + \gamma V_k(f(x,u))$$

- V_k converges to V^*: $\lim_{k \to \infty} V_k = V^*$
- Optimal policy π^* is computed at the end from $V^* \Rightarrow \pi^* = \arg\min_u \ell + \gamma V^*$

Model Free Prediction

PROBLEM: finding value function of unknown MDP by acting and observing.

SOLUTION: use MONTE CARLO METHODS.

↳ IDEA: value = mean return. Can only be applied to episodic MDP, and all episodes must end.

MC POLICY EVALUATION

Goal: learn V^π from episodes of experience under policy π: $x_1, u_1, \ell_1, x_2, u_2, \ell_2, \dots, x_k \sim \pi$

Cost-to-go is total discounted cost: $J_t = \ell_t + \gamma \ell_{t+1} + \dots + \gamma^{T-1} \ell_{t+T-1}$

Value function is expected cost-to-go under policy π: $V^\pi(x) = \mathbb{E}[J_t \mid X_t = x]$

MC policy evaluation estimates V^π as the average cost-to-go.

The first time that a state x is visited in an episode:

- Increment counter $N(x) := N(x) + 1$
- Increment total cost $C(x) := C(x) + J(x)$
- Estimate value as $V(x) = C(x)/N(x)$

By law of large numbers $V(x) \to V^\pi(x)$ as $N(x) \to \infty$

INCREMENTAL MC UPDATES

↳ IDEA: compute mean incrementally: $V_N = \frac{1}{N} \sum_{j=1}^{N} J_j = \frac{J_N + \sum_{j=1}^{N-1} J_j}{N} = V_{N-1} + \frac{1}{N}(J_N - V_{N-1})$

In a non-stationary problems it can be useful to track a running mean, i.e. forget all episodes: $V(x) = V(x) + \alpha(J - V(x))$

TEMPORAL DIFFERENCE LEARNING

It's an alternative to MC.

Estimate cost-to-go by 1-step look ahead (TD): $V(x_t) = V(x_t) + \alpha_t \underbrace{\left(\underbrace{\ell_t + \gamma V(x_t + 1)}_{\text{TD TARGET}} - V(x_t) \right)}_{\text{TD ERROR}}$

Policy evaluation is TD with $\alpha_t = 1$ and sampling all states uniformly.

PROPERTIES:

- Stochastic approximation algorithm
- If TD converges $\Rightarrow$ it must converge to V^π
- Convergence is guaranteed if step-size sequence satisfies RM conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

MC	TD
wait end of episode to know cost	Learn after every step
episodic environments	non-terminating environment
cost-to-go J_t is unbiased estimate of $V_\pi(x_t)$	target is biased estimate of $V_\pi(x_t)$
High variance, zero bias	target has lower variance than J_t
Not very sensitive to inizialization	More sensitive to initial value

EXAMPLE SLIDE 149, 150, 151

BOOTSTRAPPING: update involves an estimate
SAMPLING: updates sample an expectation

Boot. / Samp.	NO	YES
NO	Exhaustive search	DP
YES	MC	TD

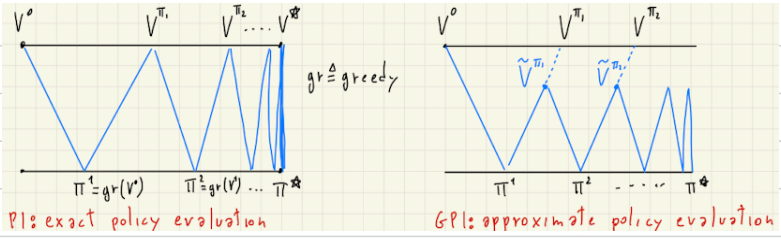
Model free control

MDP	LEARNED QUANTITIES: $V(x)$ / $Q(x,u)$	LEARNED QUANTITIES: $V(x)$/$Q(x,u)$ + $\pi(x)$
Known	Value Iteration	Policy Iteration
Unknown	Direct methods	Actor-Critic methods

Since we don't know MDP, we can't compute $V(x')$.
We have to learn Q instead of V.
$\pi(x) = \arg\min Q(x,u)$

ACTOR - CRITIC METHODS

Extension of policy iteration to unknown MDP. Policy Iteration alternates "policy evaluation" and "policy improvement".
Since we dont know MDP, policy evaluation can take infinitely many samples.
The solution is to improve policy based on approximate value function.



The value function is called "critic" because it evaluates the policy, which is called "actor", in order to improve it.
The policy used to generate transitions is tipically not the same that is evaluated and improved. The BEHAVIOR POLICY mixes small amount of exploration into TARGET POLICY.

SARSA

Use TD to learn Q from on-policy samples. After each transition (x, u, ℓ, x', u') update Q as follows:

$$\delta(Q) = \ell(x,u) + \gamma Q(x',u') - Q(x,u)$$
$$Q_{k+1}(x,u) = Q_k(x,u) + \alpha_k \delta(Q)$$

If π was fixed $\Rightarrow$ SARSA = TD

SARSA can diverge in off-policy situations.

To choose control we take policy that is greedy wrt Q. Greedy policy can be computed on the fly $\Rightarrow$ faster

- GREEDY STRATEGY: always choose control with best estimated cost. It can fail to find best action. A good learner must choose suboptimal control to explore.
- ε GREEDY STRATEGY: choose a random control with probability ε. Choose greedy control with probability $1-\varepsilon$. Simple way to ensure exploration. Probability ε can change over time (tipically decreasing to 0).

Using ε-greedy as behavior policy to ensure exploration, SARSA converges to $Q^*(x,u)$ if:

1) greedy in the limit with infinite Exploration (GLIE) policies:
   - all stat-action pairs are visited infinitely many times.
   - policy converges on greedy policy
2) Robbins-Monro sequence of step sizes: $\sum_{k=1}^{\infty} \alpha_k = \infty \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$

ON-POLICY	OFF-POLICY
e.g. SARSA	e.g. Q-learning
Learn about policy π from experience sampled from π	Learn about policy π from experience sampled from μ
Behavior policy $\approx$ Target policy	Behavior policy μ can be different from target policy π
Limited exploration $\Rightarrow$ Low sample efficiency	Higher sample efficiency

DIRECT METHODS: Q-LEARNING

Iteratively update an estimate $Q_k(x,u)$ of $Q^*(x,u)$. After observing transition (x,u,ℓ,x') update with:

$$\delta(Q) := \ell + \gamma \min_{w \in U} Q(x',w) - Q(x,u) \quad \rightarrow \text{Use } \min_w Q(x',w) \text{ instead of } Q(x',u')$$
$$Q_{k+1}(x,u) := Q_k(x,u) + \alpha_k \delta(Q_k)$$

Error $\delta(Q(x,u))$ is independent from behavior policy, which is the one that generates $u' \Rightarrow$ off-policy

Target policy is: $\pi(x) = \arg\min_u Q(x,u)$

At stochastic equilibrium, $\forall (x,u)$ visited infinitely often: $\mathbb{E}[\delta(Q)] = 0 = TQ - Q \Rightarrow Q = Q^*$

Q_k converges when appropriate learning rates are used and sufficient exploration is ensured.

PSEUDOCODE

SARSA

INPUTS: Q, ε

For $k = 0 \dots K$

  $x \leftarrow$ env.reset()

  $u \leftarrow$ EpsGreedy(Q, x, ε)

  For $t = 0 \dots N$

    $x', l \leftarrow$ env.step(u)

    $u' \leftarrow$ EpsGreedy(Q, x', ε)

    $Q^{target} \leftarrow l + \gamma\, Q[x', u']$

    $Q[x, u] \mathrel{+}= \alpha\,(Q^{target} - Q[x, u])$

    $x, u \leftarrow x', u'$

  $\varepsilon \leftarrow$ updateEpsilon(ε)

Q-LEARNING

INPUTS: Q, behavior policy π_B

For $k = 0 \dots K$

  $x \leftarrow$ env.reset()

  For $t = 0 \dots N$

    $u \leftarrow \pi_B(x)$

    $x', l \leftarrow$ env.step(u)

    $Q^{target} \leftarrow l + \gamma \min_{w} (Q[x', w])$

    $Q[x, u] \mathrel{+}= \alpha\,(Q^{target} - Q[x, u])$

    $x \leftarrow x'$

Summary

DP	TD
Policy Evaluation	TD-Learning
$V(x) = l + \gamma V(x')$	$V(x) \overset{\alpha}{\leftarrow} l + \gamma V(x')$
Policy Iteration	SARSA
$V(x) = l + \gamma V(x')$	$Q(x,u) \overset{\alpha}{\leftarrow} l + \gamma Q(x', u')$
$\pi(x) = \arg\min_u \; l + \gamma V(x'(u))$	$\pi(x) = \begin{cases} \arg\min_u Q(x,u) & \text{with prob. } 1-\varepsilon \\ \text{random} & \text{with prob. } \varepsilon \end{cases}$
Value Iteration	Q-Learning
$V(x) = \min_u \; l + \gamma V(x'(u))$	$Q(x,u) \overset{\alpha}{\leftarrow} l + \gamma \min_{u'} Q(x', u')$

Value function Approximation [continuous time]

Use function approximation to represent $V(x)$ / $Q(x,u)$

High dimensional functions:
$\hat{V}(x,w) \approx V(x)$
$\hat{Q}(x,u,w) \approx Q(x,u)$

w: parameters defining the function

$x \rightarrow [w] \rightarrow \hat{V}(x,w)$ Neural networks

$x, u \rightarrow [w] \rightarrow \hat{Q}(x,u,w)$ Differentiable Approximator: linear combination of features, Fourier basis, Polynomials, ---

$x \rightarrow [w] \rightarrow \hat{Q}(x,u_1,w)$ Non-stationary target V
$\cdots$
$\hat{Q}(x,u_m,w)$ Non i.i.d. data

VALUE APPROXIMATION BY STOCHASTIC GRADIENT DESCENT

$J_w = \mathbb{E}_\pi [(V_\pi(x) - \hat{V}(x,w))^2]$

$\Delta w = -\frac{1}{2} \alpha \nabla_w J = \alpha \mathbb{E}_\pi [(V_\pi - \hat{V}) \nabla_w \hat{V}]$

SGD samples the gradient: $\Delta w = \alpha (V_\pi(x) - \hat{V}(x,w)) \nabla_w \hat{V}(x,w)$

Expected update is equal to full gradient update.

How we get target $y = V(x)$?
- For MC is COST-TO-GO $J_t(x)$: $\Delta w = \alpha (J_t - \phi(x_t,w)) \nabla_w \phi(x_t,w)$
- For TD is TD target: $\Delta w = \alpha (r_{t+} \gamma \phi(x_{t+1},w) - \phi(x_t,w)) \nabla_w \phi(x_t,w)$

INCREMENTAL CONTROL WITH FUNCTION APPROXIMATION

Generalized Policy Iteration with approximate Action-Value Function: $\min_w \mathbb{E}_\pi [(\phi(x,u,w) - Q_\pi(x,u))^2]$

Use SGD to find local minimum.

Use ε-greedy policy to ensure exploration.

CONVERGENCE OF CONTROL ALGORITHMS

	VALUE ENCODING		
	Table Lookup	Linear	Nonlinear
MC control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-Learning	✓	✗	✗

(✓) = chatter around optimum

LIMITATIONS OF INCREMENTAL CONTROL

- Poor data efficiency
- Data is not i.i.d.

BATCH LEARNING

Instead of learning incrementally, learn after having collected a batch of training data.

Least Square prediction : $\min_{w} \sum_{t=1}^{T} (V_t - \phi(x_t, w))^2$

How to achieve it? Experience Replay:

- store data $\langle x_t, V_t \rangle$ in replay buffer
- sample minibatch from buffer
- apply SGD to update w
- converge to LS solution

DEEP Q-NETWORK

- Extension of Q-learning to continuous state space.
- Discrete control space.
- Use neural network to represent $Q(x, u, w)$.
- ε-greedy policy with ε decreasing over time.
- Use experience replay.
- Use fixed Q targets to stabilizing training
  - Store w resulting in best performance as performance may get worse during training.