CH. 1 CONVEX SETS

Convex Polyhedra

A polyhedron is a solid enclosed by poligonal faces that form a single connected region of space.

components → EDGES, FACES, VERTEX

EULER THEOREM $F + V = E + 2$ relation between edges, faces and vertex of polyhedra

A set is CONVEX if, for any two points in the set, the line segment connecting them is also contained in the set. Equation of the line:

$$\alpha p + (1+\alpha) q \qquad \alpha \in [0,1] \qquad p, q \text{ two points in the set}$$

Hyperplanes

A HYPERPLANE is the set of points that satisfy a linear equation: $a_1 x_1 + \dots + a_n x_n = b$

A HALF-SPACE is the set of points that satisfy a linear inequality: $a_1 x_1 + \dots + a_n x_n \leq b$

A SUPPORTING HYPERPLANE of set S is $f(x) = a$ if S lies within one closed half-space defined by $f(x) \leq a$ and touches S at least at one boundary point.

THEOREM OF SUPPORTING HYPERPLANES If S is convex and x_0 lies on its boundary, then there exists a supporting hyperplane containing x_0.

↳ There can be *more* than one supporting hyperplane through a boundary point.

Linear Solvability (Feasibility) Problem

Given a set of linear inequalities, determine whether there exists a feasible solution

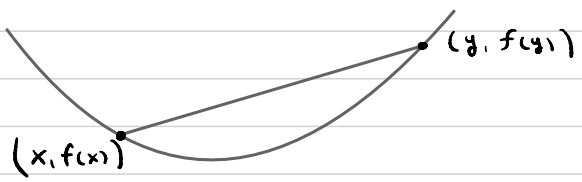
$$a_{1,1} x_1 + a_{1,2} x_2 + \dots + a_{1,n} x_n \leq b_1$$
$$\vdots$$
$$a_{p,1} x_1 + a_{2,2} x_2 + \dots + a_{2,n} x_n \leq b_p$$

This is equivalent to checking whether the corresponding polyhedron is empty.

Convex Functions

A function $f(x)$ is convex if its graph lies below the straight line between any two points: for all $x, y \in \text{dom}(f)$ and $t \in [0,1]$

$$f(tx + (1-t)y) \leq t f(x) + (1-t) f(y)$$



Convex Optimisation

A standard way of expressing convex optimisation problem is:

$$\min f(x)$$

$f_i(x) \leq 0 \quad i \in [1, m]$ convex function

$a_i^T x = b_i \quad i \in [1, P]$ equality constraints

In convex optimisation problem, any local optimus is also a GLOBAL OPTIMUS.

convex optimisation problems →
- LINEAR PROGRAMS: $\min C^T x$ s.t. $Ax \leq b$
- QUADRATIC PROGRAMS: $\min x^T Q x + C^T x \quad Ax \leq b$ with $Q \succeq 0$ pos. semidef.
  - ↳ $\min \frac{1}{2} \|Rx - d\|^2 \quad Ax \leq b$ with $c = -R^T d$
- QUADRATICALLY CONSTRAINED QUADRATIC PROGRAMS

Convex Hulls

Every set can be represented by a convex set. The CONVEX HULL of a set S, denoted conv(S) is the intersection of all convex sets containing S.

Given a set of points $A = \{x_1, \ldots, x_n\}$. Its convex hull conv(A) is given by all convex combination of points in A: $\text{conv}(A) = \left\{ \sum_{x_i \in A} \gamma_i x_i, \ \gamma_i \geq 0, \ \sum_{i=1}^{n} \gamma_i = 1 \right\}$

CH.2 GRAPH OPTIMISATION

Single-Source shortest path problem

Problem of finding the shortest paths from a source vertex v to all other vertices in a graph.

DIJKSTRA ALGORITHM Solve the problem.

APPROACH: greedy

INPUT: weighted graph $G=\{E,V\}$
source vertex $v \in V$

OUTPUT: lengths of shortest paths from v to ALL other vertices

If we store, for each node, its immediate predecessor on the minimum-cost path, we can reconstruct the optimal path.

The simplest implementation is to store vertices in an array or linked list. This yeld a running time of $O(|V|^2+|E|)$

A* search

Found the best source-goal travel. It takes care of an heuristic evaluation.

It is simple, fast and produce "good" results (can deviate from optimal path if there is not a good heuristic).

IF THE HEURISTIC $h(n)$ NEVER OVERESTIMATES THE TRUE COST, THE ALGORITHM FINDS THE OPTIMAL SOLUTION.

CH.3 DYNAMIC PROGRAMMING

Dynamic programming its a technique to solve complex optimisation problems dividing them in simpler subproblems.

A problem can be solved with DP only IF:

- OPTIMAL SUBSTRUCTURE: an optimal solution to a problem contains optimal solutions to all sub-problems.
- OVERLAPPING SUBPROBLEMS: solutions to subproblems are reused to make the solution of the bigger problem.

IMPLEMENTATION METHODS

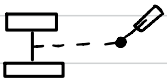
- TOP-DOWN: Start from the problem and recursively solve sub-problems. Every result is stored for a possible reuse in other sub-prob.
- BOTTOM-UP: sorted resolution of sub-problems increasing complexity.

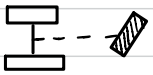
CH.4 ROBOTS : A TAXONOMY

DEXTERITY: can be defined as a robot's ability to cope with a variety of objects and actions.

STIFFNESS: ability of a body to resist deformation.

MOBILE ROBOTS
- WHEELED: a rigid body and a system of wheels that provide motion wrt the ground.
- LEGGED: multiple rigid bodies connected through revolute and prismatic joints.

Different drive Vehicle 

Tricycle Robots 

Carlike 

CONSTRAINTS : contrary to the manipulators mobile robots do not usually have a limited workspace. However they have motion constraints.
↳ Omnibot can move in ANY direction.

CH.5 MOBILE ROBOTS

The motion of mobile robots is defined through rolling and siding effects at the wheel-ground contact points. Encoder values at the link don't map to a unique pose.

There is no direct way to measure the robot's position. It must be integrated over time, depends on path taken. Leads to inaccuracies of the position (motion) estimate.
Understanding mobile robots motion starts with understanding wheel constraints placed on the robot's mobility.

Non-Holonomic Constraints

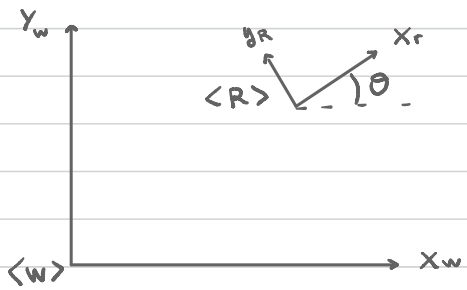
We cannot integrate the differential equations describing the motion to the final position. For a non-holonomic system, the total rotation of the wheels is not sufficient to find the final configuration. We have to know how the movement was executed in time.

$$\underbrace{S_{1R} = S_{2R},\ S_{1L} = S_{2L}}_{\text{same distance done with wheels}} \qquad \underbrace{x_1 \neq x_2,\ y_1 \neq y_2}_{\text{different final positions}}$$

Kinematic Constraints

Due to the presence of non-holonomic constraints, for mobile robots we will only deal with diff. inv. kinematics (physical => joint space).

The set of coordinates is GENERALISED if they uniquely define any possible configuration (q) of the system relative to the reference configuration.
The motion of the system, that is represented by the evolution of q over time, may be subject to constraints.



<w> world frame <R> frame

space of configurations $\in \mathbb{R}^3$: position of a point in the plane + the orientation of the rigid body.

$$T_R^w = \begin{bmatrix} R_z(\theta) & \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \\ 0 & 1 \end{bmatrix}$$

Express the kinematic of frame <R> in terms of the time derivative $\dot{x}, \dot{y}$ and $\dot{\theta}$, assuming the frame <R> is moving with a generic linear velocity v expressed in the moving frame and a generic angular velocity $\omega \hat{z}_w = \omega \hat{z}_R$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} c\theta & -s\theta \\ s\theta & c\theta \end{bmatrix} v \\ \omega \hat{z}_w \end{bmatrix}$$

Non-Holonomic Systems

Consider that a subset of coordinates q_i, with $i = 1, \dots, r < n$, is subjected to r bilateral constraints (expressed by equalities).
Constraints that are only function of the positions can be expressed as: $h(q, t) = 0$

Constraint that depend on time → RHEONOMIC.
Constraint time invariant → SCLERONOMIC $h(q) = 0$

WHR - Bilateral, Scleronomic constraints

If r constraints are defined for the system, $h(q)$ is a vector function with r entries, one for each constraint $h(q) = [h_1(q), \dots h_r(q)]$. ⇒ HOLONOMIC CONSTRAINTS.
The effect of holonomic constraints is to reduce the space of accessible configurations to a subset of $\mathbb{R}^n$ with dimension $\mathbb{R}^{n-r}$.

Dini Theorem I have an holonomic constraint expressed as: $h(q) = 0$.
If functions $h_i(q)$ are regular and Jacobian $\partial h / \partial q$ is full rank (r), so it's possible "delete" r coordinates and rewrite the configuration using only $(n-r)$ independent variables.
This theorem formalize the fact that, if constraints are integrable, you can substitute original coordinates with a reduced set of independent variables.

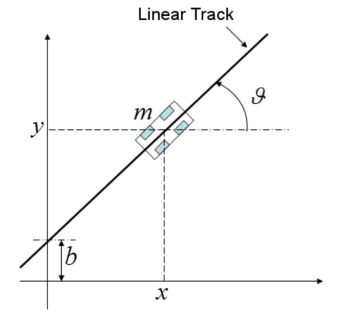
PFAFFIAN FORM A kinematic constrain connects generalized velocities $c(q, \dot{q}) = 0$. Usually constraints are linear, so they can be written as $c_i(q, \dot{q}) = a_i(q)\,\dot{q} = 0$, a scalar product between vector $a_i(q)$ and velocity vector $\dot{q}$.
Putting all constraints together we obtain $A(q)\,\dot{q} = 0$
where $A \in \mathbb{R}^{r \times n}$ collects coefficients, r = #constraints, n = #gen. coordinates

↳ Space of possible velocities $\dot{q}$ must belong to the null space $\mathcal{N}(A)$.
So we define a base $G(q)$ of the null space $\dot{q} = G(q)\,u$ u = input.

CONNECTION DINI ↔ PFAIFF

CONSTRAINT
- Holonomic: start with $h(q) = 0$. Derive in time you found a velocity constraint $\frac{\partial h}{\partial q}\dot{q} = 0$.
- Non-Holonomic: expressed directly in its velocity. $\nexists\, h(q)$ that can be derived to produce that non-holonomic constraint.

EXAMPLE : Cart Constrained on a Linear Track (Railway)



Generalised coordinates $q = [x, y, \theta]^T$.

FIRS CONSTRAINT: math desc. of railway → $h_1(q) = y - x\tan\theta_b - b = 0$

SECOND CONSTRAINT: equality between angle of track and angle of car → $h_2(q) = \theta - \theta_b = 0$ (orientation fixed by railway)

↳ $h_1(q) \Rightarrow \dot{y} - \dot{x}\tan\theta_b = 0 \qquad h_2(q) \Rightarrow \dot{\theta} = 0$

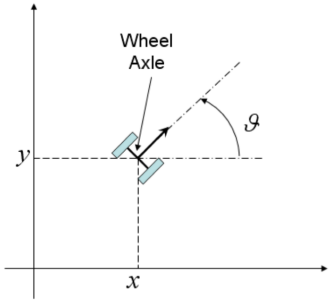
Pfaffian form: $A(q)\dot{q} = \begin{bmatrix} \sin\theta_b & -\cos\theta_b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0$

$A(q)$ has rank = 2 → null space = 1 → 1 DoF

↳ you can decide only the velocity, direction is given by railways

Kinematic model $\dot{q} = G(q)u = \begin{bmatrix} \cos\theta_b \\ \sin\theta_b \\ 0 \end{bmatrix} u$

EXAMPLE Unicycle-like Vehicle



GENERALIZED COORDINATES: $q = [x, y, \theta]^T$ x, y: center of wheel axes

KINEMATIC CONSTRAINT: $c_1(q) = \dot{x}\sin\theta - \dot{y}\cos\theta = 0$

↳ No lateral slippage

Pfaffian form: $A(q)\dot{q} = [\sin\theta \; -\cos\theta \; 0] \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0$

↳ $\dot{q} \in \mathcal{N}(A(q))$

We have 2 DoFs → 3 variables − 1 constraint

Reconstructing the base of $\mathcal{N}(A(q))$: $\dot{q} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ u_1 = linear velocity, u_2 = angular velocity

↳ $\dot{x} = u_1\cos\theta \qquad \dot{y} = u_1\sin\theta \qquad \dot{\theta} = u_2$

At the end, we don't control directly u_1 and u_2, but wheels velocity.

$$u_1 = \frac{r}{2}(\omega_r + \omega_l) \; ; \quad u_2 = \frac{r}{l}(\omega_r - \omega_l)$$

l ↳ Distance between wheels

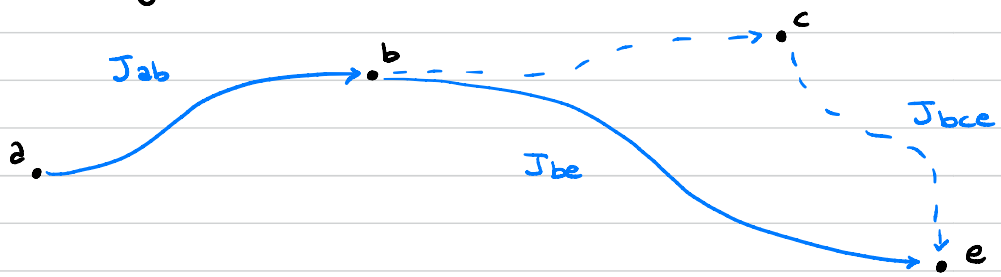
It is a non-holonomyc constraint because i cannot write $f(x, y, \theta) = cost$.

↳ Robot can reach every configuration $[x, y, \theta]$, but only with "composite trajectories" (maneuvers for parking a car).

- non-holonomyc constraints → limits velocities, not positions (unicycle) → mobile robots
- holonomyc constraints ⇒ limits <u>where</u> robot can go (railway) → manipulators

CH. 6 AN INTRODUCTION TO OPTIMAL CONTROL

The Bellman Optimality Principle An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.



The optimal choice of the path from a to b is not affected by the decisions taken to go from b to e.
If AC is the best path and includes also B, so BC must be the best path to go from B To C.

Bellman principle applies because we are considering a MULTISTAGE DECISION PROCESS. The optimal choice for the future does NOT depend on the past choices.

Generic Formula of Optimal Control Problem

Considering a dynamic system described by state equations:

$$\dot{x}(t) = f(x(t), u(t)) \quad , x(0) = x_0$$

$x(t)$: state vector
$u(t)$: control vector that we can decide
$f(\cdot)$: describes dynamic of the system

OBJECTIVE : find control function $u(t)$ that optimize (min / max) a cost function J, that is a combination of a final cost and integral cost:

$$J(x(t), u(t)) = h(x(T)) + \int_0^T g(x(t), u(t))\, dt$$

$h(x(T))$: "penalty", how far you are from destination at time T
$\int_0^T g(\dots)\, dt$: cost accumulated up To now.

→ Trova la strategia di controllo $u(t)$ (es. guidare) che, data la fisica dell'auto $f(\cdot)$, minimizza il costo totale J (es consumo benzina, tempo, ...)

The Hamilton-Jacobi-Bellman Function

We want to find a solution to the optimal control problem.

$$J(x_{(t)}, u_{(t)}) = h(x(T)) + \int_0^T g(x_{(t)}, u_{(t)})\, dt$$

with $\dot{x}_{(t)} = f(x_{(t)}, u_{(t)})$; $x(0) = x_0$; $u_{(t)} \in U$

1. DISCRETIZATION: approximation dividing time interval $[0, T]$ in N small steps of duration δ.
2. RECOURSIVE EQUATION: apply Bellman principle to write a recoursive equation back in time. Optimal cost from a state x at time k ($J^*(k\delta, x)$) is equal to the cost of actual state plus the cost of future state ($J^*((k+1)\delta, x_{k+1})$)

$$\bar{J}^*(k\delta, x) = \min_{u \in U} \{ g(x,u)\delta + \bar{J}^*((k+1)\delta,\ x + f(x,u)\delta) \}$$

3. TAYLOR EXPANSION: expand term $\bar{J}^*$ of next step with Taylor formula.
4. BACK TO CONTINUOUS: (HJB EQUATION)

$$0 = \min_{u \in U} \{ g(x,u) + \nabla_t J^*(t,x) + \nabla_x J^*(t,x)^T f(x,u) \}$$

↳ with $J^*(T, x) = h(x)$

→ In ogni singolo istante, la scelta migliore u è quella che crea un perfetto equilibrio tra:

- $g(x,u)$: il costo che paghi adesso
- $\nabla_t J^* + \nabla_x J^* f(x,u)$: la variazione del costo futuro

} come la scelta attuale influenza l'intero viaggio

If we can found a solution $V(t,x)$ to HJB equation, $u^*(t)$ it's the control that minimize expression at each time instant and the control is optimal.

The optimal control is given by:

$$u^*(t) = \arg\min_{u \in U} [g(x^*(t), u(t)) + \nabla_x J^*(t, x^*(t))^T f(x^*(t), u(t))]$$

The Pontryagin Minimum Principle

Since solve HJB equation is usually difficult, this principle offers an alternative way, giving necessary conditions for optimality.

We add new variables called co-states, $\lambda(t)$. These variables evolves following a differential equation called ADDED EQUATION:

$$\dot{\lambda}(t) = -\nabla_x g(x^*, u^*) - \nabla_x f(x^*, u^*)\lambda(t)$$

with a terminal condition: $\lambda(T) = \nabla h(x^*(T))$

So the optimal control is given by:

$$u^*(t) = \arg\min_{u} [g(x^*(t), u) + \lambda^T(t) f(x^*(t), u)], \quad \forall t \in [0, T]$$

Then we define a function, called HAMILTONIAN FUNCTION:

$$H(x, u, \lambda) = g(x, u) + \lambda^T f(x, u)$$

Maps triplet (x, u, p) to real numbers

$g(x,u)$: costo immediato

$f(x,u)$: dinamica del sistema

λ: Dice quanto è costoso essere in un certo stato

THE MINIMUM PRINCIPLE — le 3 regole che il controllo ottimo deve rispettare

Pontryagin principle says that if $u^*(t)$ it's optimal control and $x^*(t)$ is optimal trajectory, must be valid the following conditions:

(1) $u^*(t)$ must minimize $H(x, u, \lambda)$ at each time $t \in [0, T]$

$$u^*(t) = \arg\min_{u \in U} H(x^*(t), u, \lambda(t))$$

(2) Evolution of state $x^*(t)$ and co-state $\lambda(t)$ its described by Hamilton equations:

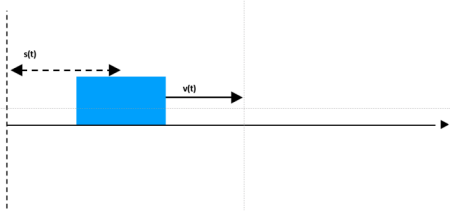
$$\dot{x}^*(t) = \nabla_\lambda H = f(x^*, u^*)$$ avanti nel tempo

$$\dot{\lambda}(t) = -\nabla_x H$$ indietro nel tempo

(3) Hamiltonian computed along optimal trajectory is costant. → controllo di coerenza

If you found a control u tha satisfies all 3 conditions, it's very likely that it's the optimal u.

EXAMPLE: Optimal Control, The problem of Minimum Time



A mass moves on a horizontal railway and it is acted on by a force $u(t) \in [-u, u]$.
Find the force function to move from a point S_0 (Velocity 0) to a point S_f (Velocity V_f) in MINIMUM TIME.

PROBLEM FORMALIZATION:

We can see the problem as the minimisation of a (virtual) integral cost index

$$\min_{u(t)} \int_0^T dt \quad \text{s.t. } S(0)=0,\ S(T)=S(f) \quad V(0)=0 \quad V(T)=V_f$$

$$\left.\begin{aligned} \frac{ds}{dt} &= V(t) \\ \frac{dV}{dt} &= \underbrace{u(t)}_{\text{la forza che causa un'accelerazione}} \end{aligned}\right\} \forall t, u \in [-1, 1]$$

(-1: max frenata; 1: max accelerazione)

OBJECTIVE: Find the combination of forces $u(t)$ that allow To reach S_f in minimum time.

(1) HAMILTONIAN FUNCTION: $H(x, u, \lambda) = 1 + \lambda_1 V + \lambda_2 u$

- 1: Il costo che vogliamo minimizzare → l'integrale è $\int_0^T (1)\, dt$
- $\lambda_1 V + \lambda_2 u$: Dinamica del sistema. λ_1 e λ_2 pesano l'importanza di ogni stato nella minimizzazione del costo totale.

(2) PONTRYAGIN PRINCIPLE: $u^*(t) = \arg\min_{u \in U} H(x, u, \lambda) = \arg\min_{u \in U} 1 + \lambda_1 V + \lambda_2 u$

In a given time instant, values of $\underbrace{V, \lambda_1, \lambda_2}_{\text{stato del sistema, fisso in un istante}}$ are fixed. We can only choose u.

To minimize $H(x, u, \lambda)$ we have to set the term λ_2.

- $\lambda_2(t)$ positive $\Rightarrow u = -1 \Rightarrow \lambda_2 u$ smallest as possible
- $\lambda_2(t)$ negative $\Rightarrow u = +1 \Rightarrow \lambda_2 u$ smallest as possible

↳ "BANG-BANG" CONTROL: we have not yet computed the solution, but we know already it is made of extreme choices: FULL THROTTLE (T), or "FULL-BRAKES" (B), as per sign of λ_2.

(3) The problem now is to find "WHEN" use T or B.
Applying Pontryagin principle, co-state variables (λ_1, λ_2) can be found solving the following differential equation:

$$\dot{\lambda}_1 = -\frac{\partial H}{\partial s} = 0 \quad \Rightarrow \lambda_1(t) = \lambda_{1,c}$$

with $\lambda_{1,c}$ and $\lambda_{2,c}$ costant

$$\dot{\lambda}_2 = -\frac{\partial H}{\partial V} = -\lambda_1 \quad \Rightarrow \lambda_2(t) = -\lambda_{1,c} t + \lambda_{2,c} \Rightarrow \text{è una retta}$$

Since the choice of u depends only on the sign of λ_2, and λ_2 is linear, can be cross the zero (and change sign) at most ONE time.
↳ Optimal strategy contain at most one change of control.

- From $t=0$ to $t=\hat{t}$ → Full Throttle (T)
- Form $t=\hat{t}$ to $t=T$ → Full brakes (B) → To "center" the destination

CH.7 THE DUBINS MANEOUVRES

PROBLEM: what is the shortest PATH to go from an initial POSITION + ORIENTATION to a final POSITION + ORIENTATION. $(x_0, y_0, \theta_0) \Longrightarrow (x_f, y_f, \theta_f)$

ASSUMPTIONS:
- constant velocity (v). So the problem switch from find the minimum time to minimum distance.
- limited curvature, minimum radius of swerved.

PONTRYAGIN PRINCIPLE: applying this principle we found that OPTIMAL STRATEGY is to use only commands for FULL THROTTLE (T) or FULL BRAKE (B).

At each time instant, the best decision is to do one of the following:

(a) L → full left
(b) R → full right
(c) S → straight

} EXTREME MANOEUVRES

DUBINS THEOREM the optimal solution is always made of 3 extreme manoeuvres. They are in 2 sets: → CSC curve, straight, curve
→ CCC curve, curve, curve

- Compute all possiple candidates paths
- measure total length of paths
- choose shortest path

Formalizing Dubins Problem

COST FUNCTION (minimum time): $\min_{\omega(\cdot)} \int_0^T dt$

CONSTRAINTS (Dynamics): $\dot{x} = v\cos\theta \quad \dot{y} = v\sin\theta \quad \dot{\theta} = \omega$

BOUNDARY CONDITIONS: $x(0) = x_0 \quad x(T) = x_f$
$y(0) = y_0 \quad y(T) = y_f$
$\theta(0) = \theta_0 \quad \theta(T) = \theta_f$

CONTROL BOUNDS: $\omega \in [-\omega_m, \omega_m], v = v_c$

we assume $v = cost$, so find minimum time is equivalent to find minimum PATH LENGTH.

COST FUNCTION: $\min_{\omega(\cdot)} \int_0^L ds$

CONSTRAINTS: $\frac{dx}{ds} = \cos\theta \quad \frac{dy}{ds} = \sin\theta \quad \frac{d\theta}{ds} = \omega$

BOUNDARY CONDITIONS: $x(0) = x_0 \quad x(L) = x_f \quad y(0) = y_0 \quad y(L) = y_f \quad \theta(0) = \theta_0 \quad \theta(L) = \theta_f$

CONTROL BOUNDS: $\omega \in [-K_m, K_m]$ (maximum curvature)

HAMILTONIAN FUNCTION: $H = \lambda_0 + \lambda_1\cos\theta + \lambda_2\sin\theta + \lambda_3 u$

$\lambda_0 = 1$ for min time/lenght problem; using u for ω

Minimum principle say that optimal control $u^*(t)$ is the one that minimize Hamiltonian at each istant.
Co-states evolve following a differential equation : $\dot{\lambda}_i = -\frac{\partial H}{\partial x_i}$. For Dubins problem λ_1, λ_2 are costant.

$$H = \lambda_0 + \rho \cos(\theta(s) - \phi) + \lambda_3 u(s) \quad \rho = \sqrt{\lambda_1^2 + \lambda_2^2} \quad \tan\phi = \frac{\lambda_2}{\lambda_1}$$

Minimize H with respect to u, wher u is vincoled to $[-k_m, k_m]$, lead to a "bang-bang" control. → $u^* = k_m$ left with min radius
↘ $u^* = -k_m$ right with min radius
Exist a singular case where $\lambda_3 = 0$. This corresponds to a straight stretch of path $u = 0$.
Optimal pats belong to 2 sets → CSC : curve, straight, curve
↘ CCC : curve, curve, curve
Equations that describes these manoeuvres can be solved in a closed form.

Dubins Curves

Solving equations for a general case is hard. The solution is to normalize problem to have initial point in $(-1,0)$ and final in $(1,0)$ 

. This is called BIPOLAR TRANSFORM, reduce parameters to only θ'_0, θ'_f, K'_{max}.
The transform is a combination of rotation, traslation and scaling.
Now the problem is only find length of 3 segments (s_1, s_2, s_2) for all possible 6 combinations. For each combinations there are closed form equations to find s_1, s_2, s_3

SUMMARY

(1) Transform to $(-1,0)$ $(1,0)$
(2) Compute s_1, s_2, s_3 and $L = s_1 + s_2 + s_3$
(3) Compare all L found
(4) Select min L
(5) Inverse transform from $(-1,0)$ $(1,0)$ to original coordinates.

Multipoint Interpolation using Dubins Curves

PROBLEM : given a set of points $P_1, \dots, P_n$ that we have to go through in order, what is the best path (shortest) that links all of them respecting some constraints.

The real problem is not to find paths, but the optimal sequence of angles $\theta_1, \dots, \theta_n$ in intermediate points.
Since the research is enormous (∞ angles for each point), the best solution is to use DYNAMIC PROGRAMMING.

1) DISCRETIZATION: instead of considering all possible angles in an intermediate point, we consider only a FINAL SET. We consider k possible angles, for example $[0, \frac{\pi}{2}, \pi, \frac{3}{2}\pi]$. This can produce an error.
2) FORWARD PASS: compute Dubins curves from $[x_1, y_1, \theta_1]$ to $[x_2, y_2, \theta_2]$ for each possible angles. Store costs in a table.
Then we consider P_3. We compute the cost from P_2 to P_3, it is the summation

cost $(P_1 - P_2)$ + cost $(P_2 - P_3)$. We store only the MINIMUM, and its the best path from P_1 to P_3.

↳ We repeat it till the final node. $C(i+1, \theta_{i+1}) = \min[C(i, \theta) + L(\dots)]$

3) BACKTRACKING: Now we know the TOTAL MINIMUM COST to go to P_n. To find the path we proceed backtracking. We answer to "Which angle θ_{n-1} have lead to this optimal solution?"

↳ Repeat till P_1.

4) REFINEMENT: discretization has introduced an error. We have to do an ITERATIVE REFINMENT.

- Doing DP algorithm with a sparse grid of angles (e.g. every 45°). This give rapid solution.
- Doing DP algorithm with a smaller grid but with only angles that are near to the previous solution
- Repeat iteratively

CH.8 COLLISION DETECTION

$$\phi(q) = \begin{cases} \text{TRUE} & \text{configuration } q \text{ is in collision} \\ \text{FALSE} & \text{configuration } q \text{ is not in collision} \end{cases}$$

Objective is to compute $\phi(q)$ the more efficent possible

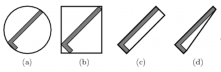
It's also useful know the distance from the obstacle. We define $dA(q)$ as the minimum distance between configuration q and set of obstacles C_{obs}.

To check collision between 2 objects we follow a 2 step strategy:

1) BROAD PHASE: using bounding volumes (3D Bounding boxes) to rapidly discard very far objects.
2) NARROW PHASE: do an accurate control on objects that have not been discarded in broad phase. This is done with more complex algorithms that consider the exact geometry of the objects.

Possible Bounding Volumes ⟶

Common choices:
- Sphere
- Axis-Aligned Bounding Box (AABB)
- Oriented Bounding Box (OBB)
- Convex Hull



HIERARCHICAL METHODS FOR EFFICIENT DETECTION

Each complex body is decomposed into a tree structure. The root of the tree has a bounding region that encloses the entire body. This region is recursively split into smaller bounding regions, creating child nodes in the tree, until the leaf nodes represent very simple geometric shapes.

Choosing a bounding region founding a trade-off between types of bounding volumes. To check for overlap between two volumes should be as computationally cheap as possible.

COLLISION CHECK PROCESS:

1. Check bounding regions of root nodes of two objects trees intersect.
2. If they overlap, the algorithm recursively checks the children of the nodes.
3. If a bounding region of a node V_1 doesn't intersect with the region of a node V_2, then non of their respective children need to be compared. This pruning drastically reduces the number of required checks.
4. When recursion reaches leaf nodes, a more precise narrow-phase algorithm is used to check for the actual intersection of the geometric primitives.

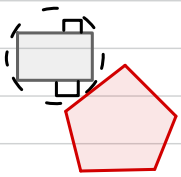
CHECKING WITH DUBINS CURVES

The narrow phase involves two key intersection tests:

1) segment-segment intersection: solved representing two line segments parametrically and solving a system of linear equations. If the determinant is zero, the segments are collinear.
2) segment-circle intersection: solved by substituting parametric equation of the line segment into the equation of the circle. This results in a second-degree equation.

POLYGONS OFFSETTING

Robot geometry can be approximate by its circumscribing circle.
The robot touches a polygon obstacle when the distance from the circle's center to the nearest edges equals the circle's radius



An effective method for collision detection in this case is POLYGON OFFSETTING:

1. Expand (offset) the obstacles by a size equal to the robot radius.
2. Treat the robot as a POINT moving along the path.
   ↳ can be implemented using CLIPPER library. [Slide 512]

CH. 9 COMBINATORIAL MOTION PLANNING

Motion planning of finding a valid, continuous path for a robot from a starting configuration to a goal configuration. This requires a model of the robot's geometry and a map of the environment, including all obstacles. The resulting plan must be ADMISSIBLE and COLLISION-FREE.

ROADMAP

Combinatorial planning is an approach that works by creating a structured representation of the free space, called roadmap.

roadmap => graph → node = safe robot configuration
→ edge = collision-free path between two nodes

↳ Finding a path becomes a simple graph search problem.

CONFIGURATION SPACE (C-SPACE)

To simplify the problem, we work configuration space instead of real-world.

Robot is simplified to a SINGLE POINT. Obstacles are "grown" or "expanded" by robot's shape.

C-space → C-obs : configurations where the robot would be in a collision
→ C-free: collision-free configurations.

Roadmap is built entirely within C-free.

METHODS TO BUILD A ROADMAP:

① Visibility Graph

Nodes of the graph are the robot's start configuration, the goal configuration and every single vertex of C-obs.

An edge is created between any 2 nodes if they are "visible" to each other, meaning a straight line connecting them doesn't pass through any C-obs.

Properties: → guarantee the shortest path. Optimal solution in terms of path lenght.
→ computationally expensive to build

② Cell Decomposition

Breaks down C-free into a collection of simple, non-overlapping regions called cells. Each cell is convex.

An adjacency graph is created where each cell becomes a node. An edge is drawn between two nodes if their corresponding cells share a boundary.

For finding a path we start finding the cell containing start conf. and the one that contains the goal. Then search the adjacency graph for a path of cells connecting the start cell to the goal cell.

Create a final path by moving through the midpoints of the shared boundaries between consecutive cells in the path.

TRAPEZOIDAL DECOMPOSITION: free space is partioned into trapezoids by drawing a vertical line upwards and downwards from every vertex of the C-obs until hits another obstacle.

PLANE SWEEP ALGORITHM: build decomposition without a huge computational cost.

The concept is to have a vertical line "sweeping" across the 2D space from left to right.

The algorithm only performs calculations when this sweep line encounters an "event point",

which are the vertices of the obstacles.

As the sweep line moves to one vertex to the next, it keeps track of the obstacle edges it is currently intersecting. At each vertex, it updates the list of intersections and uses this information to "close" old trapezoids and "open" new ones. This allow the map to be built incrementally and efficiently.

TRIANGULATION is another method for Cell Decomposition. Divides space into triangles. It connects pairs of obstacle vertices with straight lines. If a line segment connecting two vertices doesn't intersect any part of an obstacle, it's a valid "chord" that can be used to form triangles.
It is a simple and brute-force approach that checks every possible pair of vertices (computationally expensive).

GENERALIZING CELL DECOMPOSITION TO HIGHER DIMENSIONS

The concept can be extended to 3D (and even high-order) C-space.
Instead of trapezoids, you get more complex CYLINDRICAL cells.
Plane sweep algorithm can be generalized. For a 3D space, you would sweep a 2D plane along the x-axis. The "events" are now vertices, and each event triggers a 2D decomposition of the "slice" of the C-space that the plane is currently on.
For non-linear (curved) obstacles we use CYLINDRICAL ALGEBRAIC DECOMPOSITION (CAD). This is a very powerful but also higly complex method that can handle a wide variety of shapes.

APPROXIMATE CELL DECOMPOSITION

Instead of perfectly matching the geometry of the free space, this method imposes a grid over the C-space. Each cell is classified as → EMPTY: is entirely within C-free
→ FULL: is entirely within a C-obs
→ MIXED: contains a boundary between free-obs

HIERARCHICAL GRIDS start with coarse grid. All mixed cells are then subdivided into smaller cells. This process is repeated recursively until a desired solution is reached.

MAXIMUM CLEARANCE ROADMAP : Generalized Voronoi Diagrams (GVD)

Create a roadmap that prioritizes safety by staying as far away from obstacles as possible. This is achieved using GVD. The GVD is a set of points in the free space that are equidistant to two or more obstacles. GVD itself forms the roadmap. The edges of the diagram are paths that maximize the clearance (distance) from the nearest obstacles.
These paths are often not the shortest. Also, the GVD is complex.

SHORTEST PATH ROADMAP : Grazing Obstacles

It is in contrast with GVD, try to find the shortest possible path. This operation often involves grazing corners of obstacles.

The shortest path in a 2D environment with polygonal obstacles will always be a sequence of straight line connecting the start, goal, and the REFLEX VERTICES of the C-obs.

A reflex vertex is a "concave" corner of C-free, a vertex of an obstacle that you can "cut across" to find a shortcut.

Building a Roadmap: 1. nodes are the reflex vertices of C-obs

2. edges are created between pairs of reflex vertices if they are *mutually* visible and form a bitangent line.

This *method* focuses on the corner-cutting *maneuvers* needed to achieve the shortest path lenght.

CH. 10 SAMPLE BASED MOTION PLANNING

This method goes in constrast with combinatorial planning. Instead of trying to build an exact and complete map of the free space, SBMP explores the space by taking random samples.
The main advantage is that it avoids explicitly constructing the C-space obstacles.

- "BLACK BOX" COLLISION CHECKER are algorithms that rely on a powerful tool: black box collision detection module. This module's only job is to answer the question: "Is this specific robot configuration q (short path) in a collision?"

The planner doesn't need to know properties of the obstacles, it only needs to answer to this question.

GENERAL FRAMEWORK:

1. Initialize: start with an empty graph or tree, containing start configuration.
2. Sample: pick a random configuration c from the C-space.
3. Check: Use the collision checker to see if c is in C_free.
4. Connect: If the sample is valid, try to connect it to the existing graph. This is done by finding its nearest neighbors in the graph and using a local planner to see if a collision-free path exist.
5. Repeat

PROBABILISTIC ROADMAP

It'an algorithm that is useful when you need to answer many different path queries in the same environment. It works in 2 fases:

1. CONSTRUCTION (LEARNING) PHASE:
   - random samples are generated in C_free. These are the nodes of the roadmap graph.
   - for each node, look for its k nearest neighbors that are already in the graph.
   - Try to connect node-neighbor with straight lines. If it's obstacle-fee, add the edge.
2. QUERY PHASE:
   - connect q_start and q_goal to their nearest neighbors in the pre-built roadmap.
   - use graph search algorithm to find a path through the roadmap

PRM is probabilistically complete, but not guarantee to find optimal path.

RAPIDLY-EXPLORING RANDOM TREES (RRT)

Algorithm that want to find a pat quickly.

1. Pick a random sample q_rand from C-space
2. Find the node q_near that is already in the tree and is closest to q_rand.
3. From q_near, "steer" in the direction of q_rand for a small, fixed distance ε. This creates a new node q_new.
4. If the straight-line path from q_near to q_new is collision-free, and q_new to the tree with an edge connecting it to q_near.

This algorithm can explore large spaces quickly

RRT* (OPTIMAL VERSION)

Incorporates steps to RRT to find better-quality paths. Two key additions:

1. CHOOSE BEST PARENT: when q_new is created, it considers a small neighborood of nodes around it and connects to the one that offers the cheapest path back to the root of the tree.
2. REWIRING THE TREE: after connecting q_new via its best parent, it then performs a "rewiring" step. It checks if it can now provide a shorter path to any of its neighbors. If the path from the root through q_new to a neighbor is cheaper than the neighbor's current path, the tree is rewired to create this better connection.

RRT* is asimptotically optimal.

PATH SMOOTHING : the final polish

It's a post processing step used to make the path more direct and natural.

The most common technique is called SHORTCUTTING:

1. Pick 2 random points on the path found by the planner.
2. Try to connect them with a straight line.
3. If the new line is collision-free, replace the original segment of the path between those two points with this new, shorter "shortcut".
4. Repeat to improve and shorten the path.

- **Combinatorial Planners:**
  - **Pros:** They are **complete** (they will always find a solution if one exists and report failure if not).
  - **Cons:** They are generally intractable and too slow for problems with high-dimensional C-spaces and are very difficult to implement for complex scenarios.
- **Sampling-Based Planners:**
  - **Pros:** They are much more efficient in high dimensions, can handle complex robot constraints, and are simpler to implement. RRT* can even find optimal solutions.
  - **Cons:** They are only **probabilistically complete**. If a solution doesn't exist, the algorithm may run forever. Their performance can degrade in environments with very narrow passages.

CH. 11 ROBOT CONTROL

Motion planning gives us an ideal trajectory, but we have to deal with { Loc. error, Dynamics, Sensor noise }

PATH TRACKING: robot only try to follow the geometric shape of the path, regardless of time.

TRAJECTORY TRACKING: robot must follow the path and be at the correct point at the correct time. This is done by following a "virtual" robot that moves along the path according to a time schedule.

ERROR DEFINITION

$\hat{q} = (\hat{x}, \hat{y}, \hat{\theta})$: ideal pose at time t

$q = (x, y, \theta)$: actual pos at time t

Find a control law $\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = k(q(t), \hat{q}(t), \hat{v}(t), \hat{\omega}(t))$ such that $q(t)$ converges to $\hat{q}(t)$.

We epress the error in the robot's own reference frame $\Rightarrow e = (e_x, e_y, e_\theta)$. Its useful because also $v(t)$ and $\omega(t)$ are expressed in the same reference frame.

ERROR DYNAMICS: change of variables to derive the error dynamics, that describe how the error itself changes over time $\Rightarrow (\dot{e}_x, \dot{e}_y, \dot{e}_\theta)$.

↳ Problem becomes "Drive error vector to zero".

LYAPUNOV STABILITY

We can prove a system is stable if we can find a Lyapunov function $V(e)$ that acts like energy:

- $V(0) = 0$ energy is zero only at stable point
- $V(e) > 0$ energy is positive everywhere else
- $\dot{V}(e) \leq 0$ system is always losing energy

$\dot{V}$ negative semi-definite → equilibrium locally stable

$\dot{V}$ negative definite → equilibrium asimptotically stable

If we can find such a function V and design a controller that makes $\dot{V}$ negative, we have proven that our controller will work.

(1) PROPOSE a candidate $V(e)$.

(2) CALCULATE $\dot{V}(e)$.

(3) INSPECT $\dot{V}(e)$.

(4) Choose $\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}$ specifically to force $\dot{V}(e)$ to be negative.

LASALLE'S THEOREM

x_e → equilibrium point

$V: \mathbb{R}^n \to \mathbb{R}$ positive definite and continuosly differentiable function $\forall x \in \mathbb{R}^n$

ASSUME $V(x)$ is radially bounded and that its sub-level sets are bounded.

↳ Equilibrium is GLOBALLY asymptotically stable if $\dot{V}(x)$ is negative semi-definite and the only solution $x(t)$ such that $\dot{x} = f(x)$ with $\dot{V}(x) = 0$ is $x(t) = 0$ for all t.

EXAMPLE

1. Propose $V(e) = \frac{1}{2}\|e\|^2 = \frac{1}{2}e_x^2 + \frac{1}{2}e_y^2 + \frac{1}{2}e_\theta^2$ $\begin{cases} = 0 \text{ if } e = 0 \\ > 0 \text{ otherwise} \end{cases}$

2. Compute $\dot{V}(e) = e_x\dot{e}_x + \dot{e}_y e_y + \dot{e}_\theta e_\theta = e_x(v\cos\theta - \hat{v}\cos\hat{\theta}) + e_y(v\sin\theta - \hat{v}\sin\hat{\theta}) + e_\theta(\omega - \hat{\omega})$

   ↳ $\delta v = v - \hat{v}$ $\quad \delta\omega = \omega - \hat{\omega}$

3. Inspect $\dot{V}(e)$: it contains control inputs δv, $\delta\omega$

4. Choose control law: $\dot{V}(e) = -K_p e_x^2 e_y \cos(\theta - \psi)^2 - K_\theta e_\theta^2$

   K_p: gain; $\theta - \psi$: $\text{atan2}(e_y, e_x)$; K_θ: gain

Since K_p and K_θ are positive, $\dot{V}(e)$ is guaranteed to be non negative

MULTI AGENT PATH FINDING (MAPF)

The core challenge of MAPF is finding feasible joint plans for multiple agents on a graph to reach their specific goals while minimizing objective functions like Makespan or Sum of Individual Costs, all without colliding via vertex, edge, or swap conflicts. Standard approaches like A* fails in this context because the state space explodes exponentially with the number of agents, necessitating enhancements like Operator Decomposition (OD) and Simple Impedence Detection (SID) which attempt to mitigate this by serializing moves or grouping independent agents. Alternatively, Priority Planning offers a rapid, decentralized approach by planning agents sequentially based on fixed priority, but this method is fundamentally flawed as it is neither complete nor optimal.

To achieve optimality we can use Constraint Based Search (CBS) and Increasing Cost Tree Search (ICTS). CBS operates by splitting the problem into two levels : a high-level search that manages a constraint tree to resolve conflicts, similarly divides the problem but searches the space of increasing cost values, checking for valid joint paths at each cost level using Multi-Value Decision Diagram (MDD) to compactly represent all possible paths.

However, since optimal solvers do not scale to large numbers of agents, the field has moved toward bounded suboptimal and anytime algorithms. Enhanced CBS and Explicit Estimation CBS (EECBS) utilize Focal Search and Explicit Estimation Search to find solutions within a guaranteed sub-optimality bound, significantly speeding-up the high level search. For massive scalability, MAPF-LNS and X* are employed ; these quickly find an initial suboptimal solution and iteratively refine specific "neighborhoods" of the graph where conflicts occur.