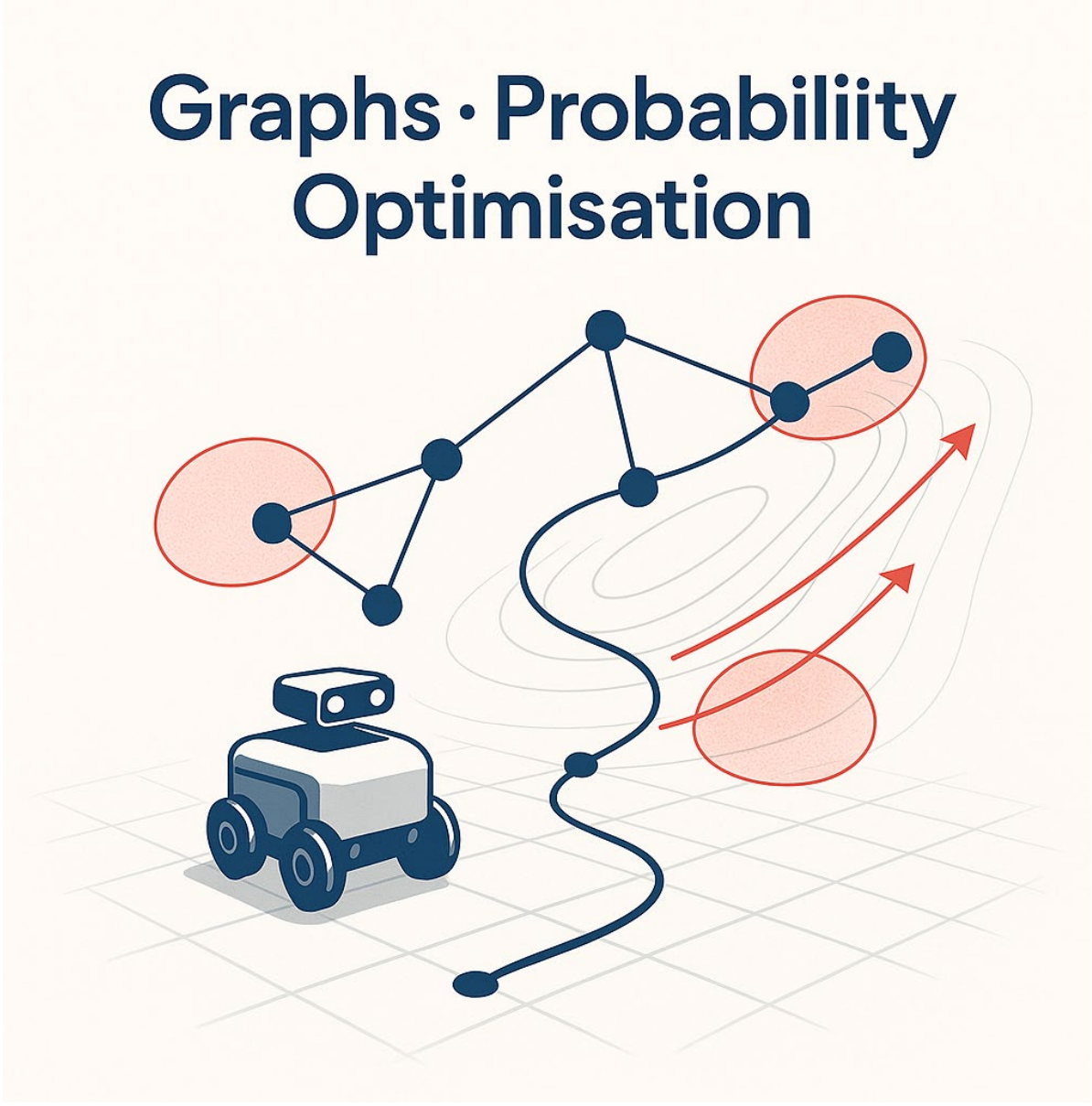
CSIC6843 – Lecture Unit 2

Mathematical Foundations for Robotics



Graph Theory, Probability & Statistics, Optimisations

Table of Contents

Unit 2: Mathematical Foundations for Robotics: Graph Theory, Probability & Optimization

Introduction

Robotics blends mechanical systems with sensing, computation, and decision-making. Three mathematical pillars support nearly every stage—from perception to motion planning to control:

Graph Theory models relationships and connectivity: robot configurations linked by feasible motions, sensor frames connected in kinematic chains, or landmarks tied together in SLAM maps.

Probability handles uncertainty: noisy sensors, imperfect actuators, and stochastic environments demand probabilistic reasoning for estimation and perception.

Optimization provides the engine to choose the best action/estimate under constraints: trajectory generation, control inputs, parameter fitting, and map reconstruction are typically cast as optimization problems.

Understanding how these three areas interlock is essential: graphs often define the structure of optimization problems; probability theory defines objective functions and constraints (e.g., maximum likelihood); optimization solves the probabilistic inference or planning task posed on a graph.

General Information

2.1. Graph Theory in Robotics

In robotics, graph theory is the use of discrete mathematical structures—graphs $G(V, E)$ consisting of vertices (nodes) and edges (links)—to model, analyse, and solve problems of motion, perception, and coordination. By mapping physical entities (robot configurations, sensor observations, tasks, or communication links) onto vertices, and relational constraints (feasible motions, spatial measurements, data flow, etc.) onto edges, many robotic problems are reduced to well-studied graph algorithms for search, optimisation, and inference. This abstraction lets us treat tasks such as localisation, path-finding, mapping, and multi-robot cooperation with rigorous guarantees on optimality, completeness, or computational complexity.

Think of a graph as a bunch of dots connected by lines. In robotics, those dots and lines let us turn messy, continuous worlds (motors moving, sensors reading, robots rolling) into something a computer can reason about quickly.

2.1.1. Building Blocks

Vertices (a.k.a. nodes = the dots)

Each dot stands for a "thing" or "situation" the robot can be in or care about, such as:

- A robot's position and orientation ("pose")
- A waypoint on a map ("go here next")
- A landmark the robot sees ("that door")
- A task ("pick up the box")

Edges (the lines)

Lines say, "you can get from this dot to that dot" or "these two things are related". Examples:

- A feasible movement the robot can make without crashing
- A joint connecting two robot parts (like elbow to forearm)
- A sensor reading that ties two poses together ("from pose A I saw landmark L, which relates to pose B")
- A measured transformation ("Pose 2 is 1 meter forward and 10° rotated from Pose 1")

2.1.2. Common Graphs in Robotics (and what they're for)

Configuration-space (C-space) graph

A discrete graph abstraction of the robot's free configuration space: each vertex is a collision-free configuration $q \mathcal{E} \mathcal{C}_{\text{free}}$; each edge is a short, locally valid motion (often a straight-line interpolation in joint space). Planning reduces to graph search for a path from the start vertex to the goal vertex

> Captures connectivity of high-dimensional spaces without storing the whole continuum; basis for grid, lattice and sampling-based planners.

Roadmaps & Visibility graphs

Roadmaps are sparse graphs that approximate $\mathcal{C}_{\text{free}}$. The classic example is the Probabilistic Roadmap (PRM): random samples become vertices, and a local planner connects nearby

samples; the method is probabilistically complete. Visibility graphs are exact planar roadmaps for polygonal environments: nodes = start, goal and obstacle vertices; edges link any two nodes that can "see" each other without intersecting obstacles, guaranteeing a shortest piece-wise-linear path.

> PRM excels in high-DOF spaces; visibility graphs give shortest paths in 2-D but hug obstacles and scale poorly in higher dimensions.

Factor graphs
A bipartite graphical model where variable nodes represent unknown states (poses, landmarks, etc.) and factor nodes encode cost/probability terms that involve subsets of those variables. Edges connect a factor to each variable it influences. Inference is performed by sparse least-squares or belief propagation.

> Unifying framework for SLAM, sensor fusion, trajectory optimisation and even task planning—exploits sparsity for real-time estimation on large problems.

Kinematic trees / graphs
A directed tree (or, for closed-chain robots, a general graph) whose vertices are rigid bodies (links) and whose edges are joints with associated transforms and limits. The root is the robot base; traversing the tree composes transforms for forward or inverse kinematics.

> Provides a structured representation for dynamics, collision checking and control of articulated manipulators and legged systems.

Topological maps
A place-graph of the environment: vertices denote distinct locations or semantic regions; edges encode reachability or qualitative connectivity (e.g., "corridor joins rooms A and B"). No metric coordinates are required.

> Lightweight navigation layer for large or dynamic spaces; supports loop-closure detection, high-level planning and human-robot dialogue ("go to the kitchen via the hallway").

These graph abstractions let roboticists trade off fidelity, computational load and the kind of reasoning required—from geometric shortest-path search to probabilistic inference and high-level semantic navigation.

The intended use will determine the choice of Graph Type.

Graph Type	What the dots mean	What the lines mean	Why we use it
Configuration-space (C-space) graph	Specific robot postures/configurations	Safe motions between them (no collisions)	Motion planning: "How do I move without hitting anything?" (PRM, RRT, etc.)
Roadmaps & Visibility graphs	Key locations or "corners" in an environment	Direct, unobstructed paths	Find shortest/safest routes through a map
Factor graphs	Unknown values we want to estimate (robot poses, landmark locations)	Probabilistic "factors"	SLAM & sensor fusion: combine noisy measurements to

		(constraints from sensors)	build maps and track pose
Kinematic trees/graphs	Robot body parts (links)	Joints that connect them	Compute how a robot arm moves, reaches, etc.
Topological maps	Big areas like rooms or corridors	Ways to move between them	High-level navigation ("From kitchen to hallway, then to lab")

2.1.3. Key Algorithms (how we use the graphs)

Robotics relies on four complementary families of graph algorithms. Together they let us search, sample, optimise, and analyse the richly–structured graphs introduced in Sections 2.1.1 – 2.1.2.

Family	Core Ideas	Typical Algorithms	When a Roboticist Reaches for It
Discrete graph search	Treat motion or information flow as a weighted path over an explicit graph	Breadth/Depth-First Search, Dijkstra, A* (with admissible heuristics), k-best path variants (Yen, Eppstein)	Mobile robot following a 2-D occupancy grid in a hospital; manipulator routing a cable through an already-meshed CAD model
Sampling-based motion planning	Approximate high-DOF configuration space with a road-map or tree built from random, collision-free samples	PRM / PRM*, Lazy-PRM, RRT, RRT*, BIT*, FMT*	7-DOF industrial arm inserting a part inside an engine block; UAV exploring cluttered forest where an explicit grid would explode in size
Graph-based optimisation & SLAM	Write state estimation as a factor (pose) graph and solve a sparse, non-linear least-squares problem	Gauss-Newton, Levenberg–Marquardt, iSAM2 incremental updates, g2o/GTSAM toolkits	Autonomous car revisiting a downtown block and performing loop-closure to eliminate accumulated drift
Structural analysis & maintenance	Maintain global properties that guarantee performance or safety	Connected-components, incremental MST, cycle detection, cut-estimation	Multi-robot team monitoring whether the communication graph stays connected while agents spread out

Discrete Graph Search

- BFS / DFS answer reachability questions—useful for quickly checking whether a pick-and-place task is even possible before expensive planning.
- Dijkstra guarantees the shortest cost path but explores uniformly; A* adds an informed heuristic (e.g. Euclidean distance) that directs the search front, giving real-time performance for warehouse AMRs.
- k-best (Yen/Eppstein) produce multiple alternatives so a scheduler can swap routes on-the-fly when humans step onto the shop-floor.

Example: A domestic vacuum forms a 2-D grid of the floor, weighs each cell by how dirty it is, then runs A to find the least-time path that still hits the dirtiest patches first.*

Sampling-based Motion Planners

Why sample? In a 6-DOF arm, even a coarse 20 samples/axis grid would explode to 64 million nodes. Road-map/tree planners avoid this curse:

- PRM family (multi-query): build once, reuse. PRM* adds asymptotic optimality; Lazy-PRM delays expensive collision checks until a query arrives.
- RRT family (single-query): grow a tree from the start; RRT* rewires to converge on the optimum; BIT* and FMT* batch samples and reuse a heuristic like A*, blending search and sampling.

Example: An anthropomorphic arm performing sand-blasting inside a ship hull seeds RRT with arm configurations biased toward the end-effector's target patch, enabling a solution in seconds instead of minutes.*

Practical tips

1. Tune the connection radius to stay probabilistically complete yet sparse.
2. Use Gaussian or bridge sampling near obstacle boundaries when the space is dominated by narrow passages (e.g. threading wires).

Graph-based Optimisation (Factor Graphs)

In SLAM and sensor fusion we convert every noisy measurement into a factor that glues together a subset of unknown variables (poses, landmarks, calibration parameters). The resulting least-squares objective is the same sparse structure exploited by trajectory optimisation.

- Batch solvers (Gauss-Newton, Levenberg–Marquardt) yield globally consistent maps after a loop closure.
- Incremental solvers (iSAM2) update an existing QR/Cholesky factorisation in per new edge—crucial for real-time autonomy.

Example: A drone mapping a cave uses visual-inertial odometry to add edges every 10 milliseconds*; iSAM2 updates in ~1* milliseconds*, keeping CPU free for control.*

Structural Graph Operations

Even with optimal paths or accurate maps, graph health matters:

- Connectivity tests ensure a road-map contains at least one feasible path before A* starts—saving costly search on a disconnected graph.
- Minimum Spanning Tree (MST) gives the lightest tether-cable layout for an underwater ROV swarm while guaranteeing every vehicle has power.
- Cycle detection triggers loop-closure candidates in SLAM; adding an edge that closes a long cycle greatly reduces drift.

A small set of incremental algorithms keeps these properties updated as nodes/edges are added or removed—mirroring how the robot's internal world model evolves during operation.

Choosing the Right Toolbox

- Low-DOF / known map → discrete search.
- High-DOF / implicit obstacles → sampling-based planners.
- Noisy sensors / drifting odometry → factor-graph optimisation.
- Dynamic teams / limited bandwidth → structural maintenance (connectivity, MST).

The following flowchart (see Fig. 2-4 in Section 2.1.4) summarises these decision points and highlights where probability (uncertainty weighting) and optimisation (cost functions) plug into each algorithmic family.

Key Take-away: Graph algorithms are not one-size-fits-all; selecting and combining the right family for a given task—often stacking two or more in a pipeline—turns an otherwise intractable geometric or probabilistic problem into something a robot can solve on-board and in real-time.

2.1.4. Why Bother With Graphs?

Robots inhabit a continuum—infinite poses, joint angles, and sensor readings. Graphs carve this space into a finite, algorithm-friendly scaffold on which we can reason, learn, and optimise. Four recurring pay-offs motivate their ubiquity:

Compression without blindness

Discretise just enough: A 2-D grid map with millions of cells collapses to a few thousand way-points in a PRM; a 6-DOF manipulation task reduces to tens of keyframes. The robot still "covers" the space, but with orders-of-magnitude fewer states to explore.

Plug-and-play modularity

The same vertex/edge container can host geometry (collision-free edges), probability (weighted factors), energy cost, or even semantic labels. Down-stream modules—search, SLAM, task planning—swap in specialised algorithms without rewriting the world model.

Provable efficiency

Mature graph algorithms come with hard runtime bounds:

- A* returns the optimal path in $\mathcal{O}(\mathcal{E} \log \mathcal{V})$
- Incremental connectivity/MST updates cost near-constant time
- iSAM2 keeps SLAM updates around $\mathcal{O}(\log n)$ instead of$\mathcal{O}(n^3)$. Real-time autonomy hinges on such guarantees.

Visualisation & debugging super-powers

Plotting a factor graph, PRM, or comms network reveals missing loop closures, isolated components, or overloaded links—making graphs an indispensable explainability layer in field robotics.

Big picture: Think of a graph as the data-structure glue that lets probability and optimisation talk to geometry. Figure 2-4 visualises this triad: continuous world → graph abstraction → probabilistic inference → optimal decision.

Key Take-away: “Graph first” often turns an intractable calculus problem into a linear-algebra one the robot can solve at 100 Hz on embedded hardware. Whenever complexity creeps in, ask: Can I draw the dots and lines?

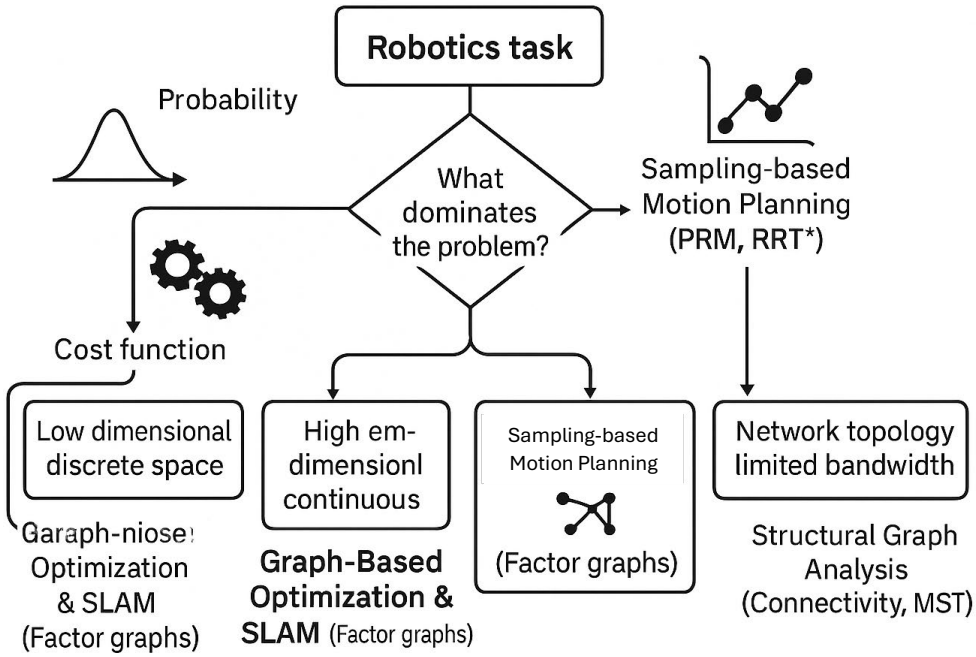


Figure 2-4.

Supporting Information

Abbreviation	Full term	Short description
2-D	Two-dimensional	Refers to planar (x–y) environments or problems constrained to two spatial dimensions.
*A**	A-star Search	Dijkstra's algorithm accelerated with an admissible heuristic that guides the search toward the goal.
BFS	Breadth-First Search	Explores a graph level-by-level to answer reachability or discover the shortest un-weighted path.
*BIT**	Batch Informed Trees	Heuristic-guided, batch version of RRT* that focuses samples where better paths are likely to lie.
C-space	Configuration space	The set of all possible robot configurations; often written $\mathcal{C}$. Its collision-free subset is $\mathcal{C}_{\text{free}}$.
DFS	Depth-First Search	Follows each branch as deep as possible before backtracking; handy for exhaustive exploration or cycle detection.
DOF	Degrees of Freedom	Independent coordinates needed to specify a robot's configuration (e.g., a 6-DOF manipulator has six joint variables).
*FMT**	Fast Marching Tree	Builds a tree using a cost-to-come wave-front (like Dijkstra) while retaining sampling-based scalability.
g2o	General Graph Optimisation (open-source library)	C++ framework that solves the large sparse nonlinear least-squares problems arising in SLAM and bundle adjustment.
iSAM	Incremental Smoothing And Mapping	Family of algorithms (e.g., iSAM2) that update the SLAM factor graph in real time using sparse matrix re-factorisation.
MST	Minimum Spanning Tree	Subset of edges that connects every vertex with the minimum possible total weight—useful for wiring, communication backbones, etc.
PRM	Probabilistic Roadmap	Sampling-based planner that builds a sparse roadmap of $\mathcal{C}_{\text{free}}$ by randomly sampling vertices and connecting nearby ones.
*PRM**	Probabilistic Roadmap	Sampling-based planner that builds a sparse roadmap of collision-free configurations for multi-query motion planning.
RRT	Rapidly-exploring Random Tree	Grows a tree from the start configuration toward random samples; good for single-query, high-DOF problems.
*RRT**	Optimal Rapidly-exploring Random Tree	As RRT but rewires the tree to asymptotically converge on the optimal path cost.
SLAM	Simultaneous Localisation and Mapping	Estimation problem in which a robot builds a map of an unknown environment while concurrently determining its own pose within that map.
SLAM	Simultaneous Localisation and Mapping	Jointly estimates the robot's trajectory and a map; factor-graph optimisation is the state-of-the-art formulation.

Key	**Meaning**
Dot	Situation/thing.
Line	Relation/movement.
C-space graph	“Where can I move safely?”
Factor graph	“How do my noisy measurements fit together?”
Kinematic graph	“How are my joints linked?”
Topological map	“Which room leads to which?”
A, Dijkstra*	Shortest path.
PRM/RRT	Explore big spaces by sampling.
Pose graph optimization	Tidy up the whole map after collecting sensor data.
Connectivity/loops/trees	Structural properties that affect planning and optimization.

2.2. Probability & Statistics in Robotics

(the "uncertainty toolbox")

Robots never see the world exactly as it is; wheels slip, cameras glare, and GPS fades in the city canyon. Probability and statistics give us a principled way to represent what we don't know, update those beliefs as fresh data arrives, and still take sensible actions in real time.

2.2.1. Fundamental concepts

- Random variable – any quantity we treat as uncertain (e.g., the true distance just travelled).
- Probability distribution – our current story about where that variable might land.
- Expectation & covariance – the "centre" of the story and how wide it spreads; covariance also tells us which uncertainties move together (camera yaw error often correlates with depth error).
- Conditional probability – given a new observation, how does that story change?

Illustration: A warehouse robot thinks the pallet is ~2 m away ± 5 cm. An ultrasonic ping confirms it is closer; the posterior shrinks to ±2 cm, automatically shifting the planned arm motion.

2.2.2. Bayesian belief updates

Bayesian reasoning simply says: start with a prior guess, compare it against new evidence, and keep what remains plausible. In practice, the robot keeps a compact numeric representation of that belief (a mean + covariance matrix, or a cloud of particles) and refreshes it hundreds of times per second as sensors stream in.

2.2.3. Filtering – estimating "right now"

Filter family	When it shines	Example
Kalman (linear-Gaussian)	Precise, nearly linear motion (industrial gantry)	Laser tracker plus encoders keeps a milling head within 0.1 mm.
Extended / Unscented KF	Mildly nonlinear dynamics (differential-drive rover)	EKF fuses wheel odometry with IMU yaw.
Particle filter	Highly nonlinear or multimodal beliefs	"Kidnapped robot" in RoboCup localises from scratch with 2 000 particles.

2.2.4. Smoothing and factor graphs

If the robot can afford to look backwards, it re-optimises an entire window of past states, blending future evidence to clean up earlier drift. Factor-graph tooling (e.g., GTSAM) turns this into a sparse optimisation problem that runs in real time even on embedded CPUs.

2.2.5. Reasoning over sequences

- Markov chains & HMMs – predict the next discrete scene ("door, corridor, lab") from Wi-Fi signatures.
- Gaussian processes – map crop height across a field with confidence bands, telling the drone where more sampling is worthwhile.

2.2.6. Communicating confidence

Robots visualise ellipses around pose estimates, publish 95 % intervals to downstream planners, and annotate maps with colour-coded “heat” for uncertainty so operators can quickly spot weak areas.

2.2.7. Sensor & motion noise models

Every estimator is only as honest as the noise models it ingests. Teams routinely calibrate wheel-slip variances on different floor materials or build per-pixel depth-sensor error tables learned from ground-truth rigs.

2.2.8. Acting while uncertain

Planning in belief space (POMDPs) means balancing doing the job with gathering information. A self-driving car might slow briefly to let its lidar finish a full sweep before committing to a tight merge.

Supporting Information

Key	**Meaning**
Random variable	Uncertain number.
Bayes	Prior × Likelihood → Posterior.
Filter	Estimate now (KF, EKF, UKF, Particle).
Smoother	Re-estimate past with future data (factor graphs).
Markov / HMM	Probabilistic state sequences; HMM hides the state.
GP	Predict a smooth function with uncertainty bars.
Covariance ellipse	Picture of how unsure you are.
Motion/Sensor models	Noise models that drive all inference.
POMDP	Partially Observable Markov Decision Process (POMDP). Act while uncertain; plan over beliefs.

2.3. Optimization in Robotics

(the "decision engine")

Once we have a structured belief about the world, optimisation algorithms pick the best trajectory, control input, or parameter set—subject to physics, safety, and energy limits.

2.3.1. Problem framing

1. Objective – a single scalar we aim to minimise or maximise (time-to-goal, energy, tracking error).
2. Constraints – keep the answer physically feasible (dynamics, joint limits, collision clearance, battery budget).
3. Convex vs. non-convex – convex problems guarantee a single global optimum (great for real-time MPC); non-convex ones abound in robotics, so we often relax, warm-start, or solve a series of convex surrogates.

2.3.2. Common optimisation flavours

Class	Typical use	Field example
Least-squares	Map & parameter estimation	SLAM bundle-adjusts 3 000 keyframes overnight.
Quadratic Program (QP)	Model Predictive Control	Biped keeps balance by solving a 20-variable QP at 200 Hz.
Nonlinear Program (NLP)	Aggressive trajectory generation	Racing drone plans a minimum-snap path through gates.
Mixed-Integer Program (MIP)	Contact sequencing / logic	Humanoid chooses which foot lands next on uneven rubble.

2.3.3. Solver toolkit

- Gradient descent – the Swiss-army knife; good initial pass.
- Gauss–Newton / Levenberg–Marquardt – lightning-fast for least-squares SLAM.
- Sequential Quadratic Programming – handles complex constraints in manipulation.
- Interior-point & ADMM – scalable to hundreds of robots cooperating over Wi-Fi.
- Realtime tricks – warm-start with yesterday's plan; pre-factor dynamics Jacobians; offload heavy matrix algebra to the GPU.

2.3.4. Bridges to probability

Maximum-likelihood and MAP estimation are just optimisation problems where the cost is the negative log-likelihood. Covariance matrices from Section 2.2 naturally become weights inside the objective, linking the two pillars.

2.3.5. Bridges to graphs

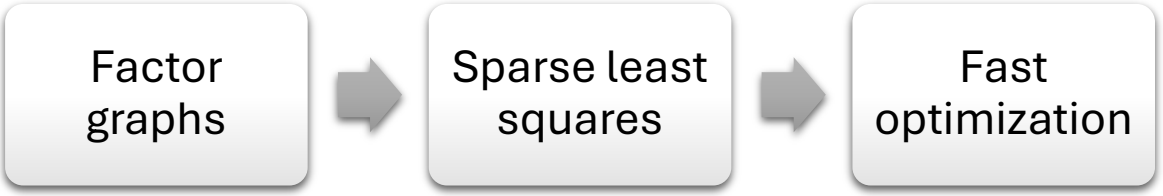
Factor graphs make many optimisation tasks sparse: each variable only touches a few neighbours, so linear-algebra kernels exploit that sparsity for speed. Pose-graph SLAM is the canonical example.

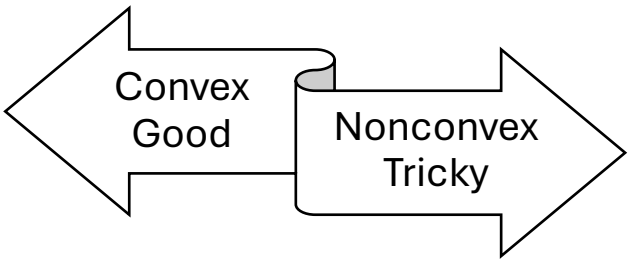
Take-away:

Probability tells the robot what might be true, optimisation decides what to do next, and both lean on the graph structures introduced in Section 2.1. Together they form a tight feedback loop that lets modern robots operate safely, efficiently, and autonomously in the messy real world.

Supporting Information

Key	**Meaning**
Cost	What we're shrinking (time, error, energy).
Constraints	Must-follow rules (physics, collisions).
Common problem flavours	Least-squares, QP, NLP, MIP.
Popular solvers	Gauss–Newton, SQP, interior-point, ADMM.
ML/MAP	Optimization view of probability.





Key Highlights — How the three pillars work together

Robotics only sings when structure, uncertainty, and decision-making talk to one another:

- Graphs – the skeleton. Nodes capture states, poses, tasks; edges encode motions, constraints, or data links. They give every problem a tidy, discrete backbone.
- Probability – the fog meter. Overlays each node/edge with a measure of confidence so the robot knows where it might be wrong.
- Optimisation – the clean-up / decision engine. Drives the whole structure toward the most self-consistent map, path, or control action.

Together they form a loop: graphs supply structure to the likelihoods and costs, probability weights those costs by uncertainty, and optimisation searches that structured space for the best answer.

Canonical showcase — SLAM

A mobile robot in a factory corridor:

1. Graph: Builds a factor-graph whose nodes are time-stamped poses & landmarks; edges are sensor constraints.
2. Probability: Attaches noise models (e.g. lidar 3 m ± 0.1 m), turning raw residuals into weighted errors.
3. Optimisation: Runs sparse nonlinear least-squares to slide every pose/landmark until the residuals shrink, giving a globally consistent map and trajectory.

The result is a centimetre-level map computed on-board in real time.

Motion planning in a cluttered warehouse

Sample–connect–optimise:

- Graph: PRM/RRT samples safe configurations and knits them into edges.
- Probability: Adds chance constraints so the chosen route keeps collision risk below, say, 1 %.
- Optimisation: Minimises travel time and jerk while respecting actuator limits.

A forklift can now whisk pallets quickly yet conservatively between congested aisles.

Model-Predictive Control (MPC) on the fly

Every 10 milliseconds the controller:

- Solves a tiny QP/NLP to find steering, throttle, or joint torques under dynamic constraints.
- Folds in process and measurement noise so actions remain stable on slippery floors.
- Decomposes across robots if a swarm must co-ordinate (distributed optimisation on sub-graphs).

Computational survival kit

Robots live on tight CPUs and stricter deadlines:

- Lean on sparse Cholesky/QR to smash huge but mostly empty matrices.

- Re-use yesterday's solution as a warm start and keep latency bounded.

Toolbox quick-reference

When building your own stack, proven libraries save months:

Pillar	Go-to libraries	Typical use-case
Graph / SLAM	g2o, iSAM2, GTSAM	Incremental factor-graph optimisation on a rover
Planning	OMPL	RRT*/PRM planning for 6-DOF manipulators
General optimisation	Ceres, IPOPT, OSQP	Bundle-adjustment, nonlinear trajectory design, QP-based MPC

What the maths can promise

- Probabilistic completeness: keep sampling and a path (if one exists) will be found.
- Estimator consistency: EKF is optimal only under linear-Gaussian assumptions. Break them and guarantees vanish.
- Convergence rates: from linear to quadratic — faster shrinkage per iteration but often heavier algebra.

Emerging currents you'll meet in graduate papers

- Learning-informed optimisation: neural nets warm-start the solver.
- Graph neural networks: operate directly on factor-graphs for loop-closure detection.
- Probabilistic programming: specify sensor models once, let inference handle the algebra.
- Real-time convexification: turn gnarly problems into rapid sequences of convex sub-problems on-board the bot.

Take-away

Graphs give you a scaffold, probability tells you how shaky that scaffold is, and optimisation tightens the bolts. Master the loop and your robot will reason, plan, and act with the confidence of a seasoned field engineer—even when the world throws glare, grit, and Gaussian noise its way.

Supporting Information

Key	**Meaning**
SLAM	Factor graph + Noisy sensors + Least-squares solve.
Planning	Nodes/edges + Risk limits + Smooth, Efficient paths.
Control/MPC	Fast optimization with noisy models.
Speed matters	Sparse math, Warm starts.
Libraries	• Graphs: g2o/iSAM2/GTSAM. • Planning: OMPL. • Optimization: Ceres/IPOPT/OSQP.
Guarantees	• Completeness (planners). • Optimality (filters under assumptions). • Convergence rates (optimizers).
Trends	learning helps optimize faster; new math makes hard problems real-time friendly.