Lecture 7, Jan 19, 2026

Introduction to Planning and Graphs

- In general planning involves finding a path between two configuration states, i.e. answering queries about the connectivity of some space, subject to constraints and optimality criteria
- A classic manipulator planning approach was to find the entire allowed configuration space by mapping all the obstacles into the configuration space, and finding a path in space through only allowed regions
  - This is expensive and only feasible for static situations such as factory robots, which could be engineered in advance
  - Mobile robots cannot use this approach since their environments are much larger and typically not known in advance
  - The physical space is in general much easier to define, but the configuration space is much easier to plan in
- For simplicity we assume that our localization and mapping are perfect; planning under uncertainty is much more computationally expensive
- Planning often utilizes a two-tiered approach: a strategic/high-level (global) planner, planning an optimal route based on a priori knowledge of the environment (e.g. satellite image and GPS), and a tactical/low-level planner that handles real-time local planning for obstacle avoidance (reactive methods)
  - This leads to a standard workflow where we first compute a simple path, smooth the path to account for kinematics, design a trajectory to account for dynamics (i.e. adding a velocity profile), and designing a controller to track the trajectory

Reactive Planning

- *Potential fields* is one of the simplest approaches for planning, based on finding the minimum of a potential field
  - The target potential attracts towards the goal: $V_{att}(q) = K_{att}\rho(q, q^g)^2$
    - q is the current configuration, q^g is the goal configuration
    - $\rho(q, q^g)^2$ measures the distance between the current configuration and the goal, often in physical space
  - Obstacle potentials repel the vehicle: $V_{rep}(q) = K_{rep} \sum_{i=1}^{n} \frac{1}{\rho(q, O^i)^2}$
    - $\rho(q, O^i)$ computes the shortest distance between q and obstacle i (more precisely the closest point on the obstacle surface)
    - We often impose a maximum range of influence so that outside of this distance the obstacle potential is zero
  - The total potential is the combination of the two fields: $V(x_t) = V_{att}(x_t) + V_{rep}(x_t)$
  - To find the path, use a gradient descent approach; at each step we move in the direction of steepest descent of the potential field, which can be found by taking the gradient $-\nabla V$
  - The path often looks like a straight movement towards the goal until the robot gets within the region of influence of an obstacle, and then it's pushed around the obstacle
  - The major weakness of potential fields is the existence of local minima in the potential function in complex scenarios, which the robot can get stuck in and cannot recover from
  - Potential fields are also non-optimal and do not consider dynamic constraints
- *Extended potential fields* adapt the method specifically for driving robots by incorporating the vehicle heading
  - This adds a rotation potential, with a dependence on bearing to obstacles so that the robot turns away from obstacles and also diminishes the influence of obstacles behind the robot
- *Trajectory rollout* is a very basic form of MPC that considers a discrete number of possible trajectories in a short time window
  - Consider n different inputs to apply, e.g. holding velocity constant and considering n different rotation rates
  - For each of the inputs, propagate the trajectory for some time T using a vehicle model, and check

each trajectory for collisions
  - Out of the valid trajectories, score on criteria such as progress to goal, distance to obstacles, similarity to previous choice, etc.
  - If computing trajectories takes time, we need to predict where the starting point is by the time we're done planning, using the current inputs
  - Kinematic and dynamic constraints can be considered by constraining the input choices based on the current state at each step, e.g. restricting the possible changes to angular and translational velocity in one step
  - As a primarily local planner, trajectory rollout can still get stuck due to the very limited planning window

Global Planning

- The environment can be represented as a regular grid of traversable vs. non-traversable locations, producing an *occupancy map*, or irregularly sampled locations connected by traversibility, producing a *probabilistic roadmap*
  - PRMs have the advantage of being able to operate in physical space or configuration space, as long as we have a way to check whether two configurations are connected
- Many of these methods lead to graphs, with nodes representing allowable configurations and edges connecting them; once the graph is built, the problem simply becomes a shortest path search on a graph
- Graph-based planners should satisfy some properties:
  1. Completeness: In finite time, the planner should either produce a path or conclude that no path exists, i.e. a complete planner never misses a valid path
     - For grid-based planning, a *resolution complete* planner is guaranteed to find a path, if one exists, if the resolution of the grid is fine enough
     - For probabilistic planning, a *probabilistically complete* planner has a failure probability asymptotically approaching zero as more work is performed
  2. Optimality: The planner should report the best (lowest cost) path
     - A planner is *anytime optimal* if, as more work is performed, progressively better paths are found, i.e. we can cut off the planning early to find a feasible but less optimal path
  3. Efficiency: The planner should find the solution in a timely manner
- The planning task involves finding a sequence of actions that take us from the initial state x_I to some final state within the goal set X_G
  - We can use a generic forwards-search algorithm to find the path
    * At each step, take the next state x from the queue, mark it as visited, discover all unvisited nodes reachable from x, and add each reachable unvisited node to the queue
  - States that have been encountered but not visited are kept in a queue, and the prioritization of this queue leads to different search algorithms:
    * LIFO (stack): depth-first search
    * FIFO (queue): breadth-first search
    * Cost-to-come (i.e. cost to get to the node): Dijkstra's algorithm
    * Heuristic-guided cost function: A*
    * Estimated cost function: suboptimal best-first