Path Planning & Collision Checking

Path planning is "inverse kinematics over time". Goal:

- Produce a trajectory that is
  - Kinematically feasible
  - Dynamically safe \(\leftarrow\) we'll mostly ignore dynamics
  - Collision free
  - Achieves desired state (e.g., position, or position at time \(t\), or position & velocity at time \(t\)).

Collision tests at particular states

Basic idea: decompose robot & world into simple geometric primitives.
- Triangles if you must, spheres if you can, (+ planes)

\[
\begin{array}{ccc}
\text{\includegraphics[width=0.3\textwidth]{triangle}} & \Rightarrow & \text{\includegraphics[width=0.3\textwidth]{spheres}}
\end{array}
\]

Given state (joint angles), use full kinematics to compute position of every geometric primitive. Does any pair collide?

Sphere filling is an engineering trade-off

- Missing "stuff", but few spheres.
- Missing less, but many spheres
- Includes too much stuff

Configuration Space

Set of legal positions. Can be high-D (6DoF for our arm)

Is Collision? \((\theta_1, \theta_2, \theta_3, \theta_4, \ldots)\)

In low-D (e.g., mobile robot), can pre-compute config space

Is Collision? \(\Leftrightarrow\) look-up table,
Collision Checking Paths

Is it safe to go from \([\theta_1, \theta_2, \theta_3, \ldots]\) to \([\theta'_1, \theta'_2, \theta'_3, \ldots]\)?

Approach:
- what is the path? (linear interpolation often used)
- sample path for collisions

Finding paths in configuration space

In low-D, brute-force search, e.g. wavefront (or A*)

In high-D, often randomized search (e.g. RRT)

Wavefront+

- Let \(W\) be the set of goal positions.
- Let \(L = 1\)
- Until path-cost (shortest) known:
  - Let \(W\) be the set of positions one step away from \(W\),
  - Set path-cost(\(W\)) = \(L\).
  - \(L++\)

Generalization:
Each state has "temporal cost"
Use heap to expand burst-cost path.
Consider a tree of actions

- reject edges that are bad
  - collisions
  - dynamically infeasible

Note: we're considering (explicitly) action sequences: easy to estimate passage of time,
  \[ \Rightarrow \text{"easy" to handle moving obstacles.}\]
  Can test for collisions as \( f \) (time).

Basic algorithm

1. Pick a node in the tree
2. Expand it with an action

直到路径找到

RRT

1. Pick a random point in state space
2. Try to grow a current node (the closest?)
   towards that point.
  \[ \Rightarrow \text{fills space rapidly (good explanation)} \]

完成？
最优？
实用？