Non-holonomic planning

Jane Li

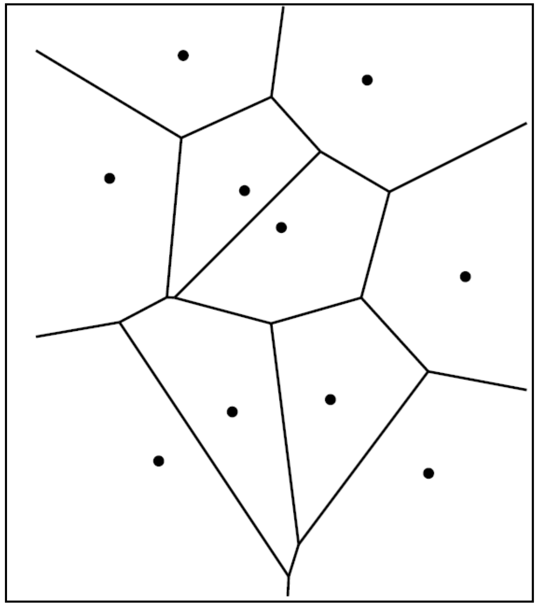
Assistant Professor
Mechanical Engineering Department, Robotic Engineering Program
Worcester Polytechnic Institute

Quiz (10 pts)

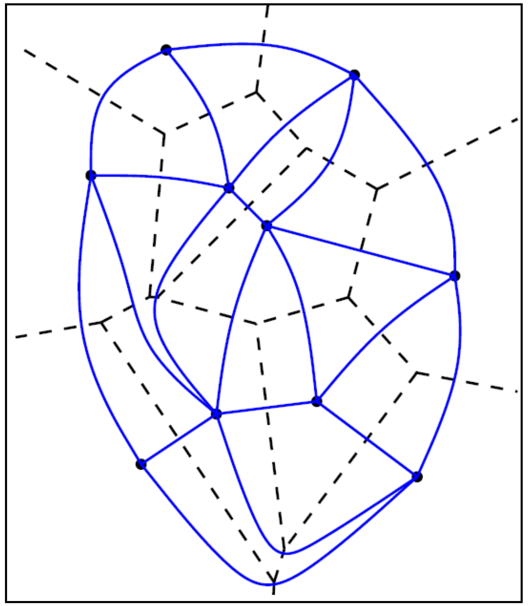
- (3 pts) How to generate Delaunay Triangulation?
- (3 pts) Explain the difference between AABBs and OBBs
- (4 pts) Explain how to check collision using BVH

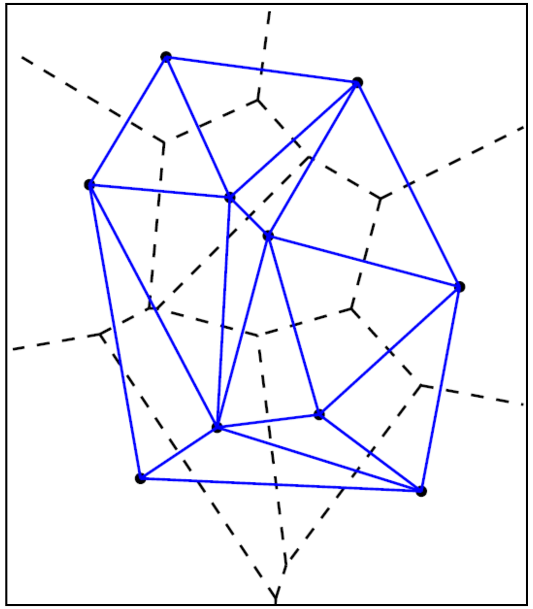
Delaunay Triangulation

- Goal – Avoid sliver triangle
  - Find the dual graph of Voronoi graph



Voronoi Graph

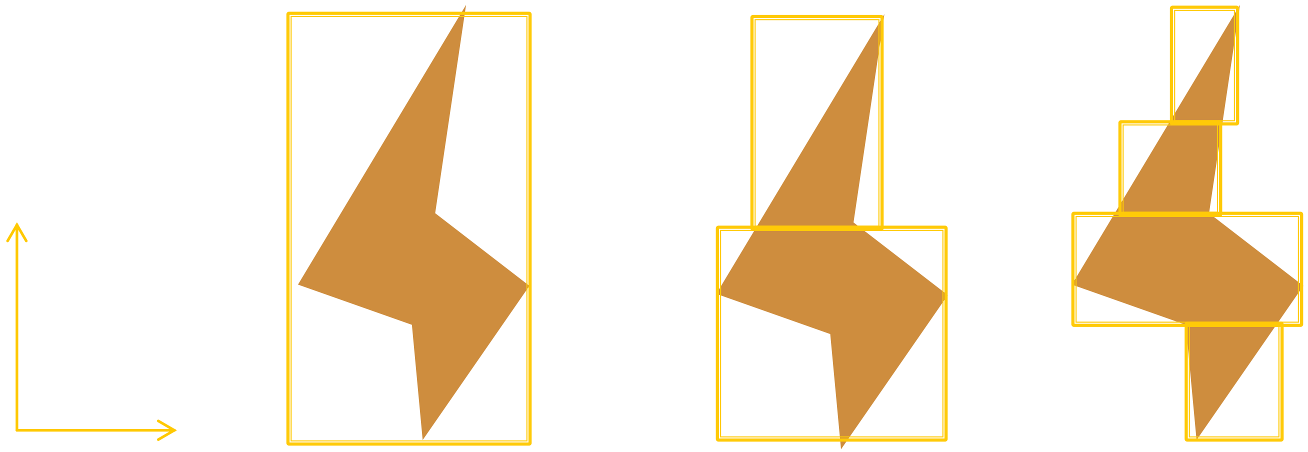




Delaunay Graph

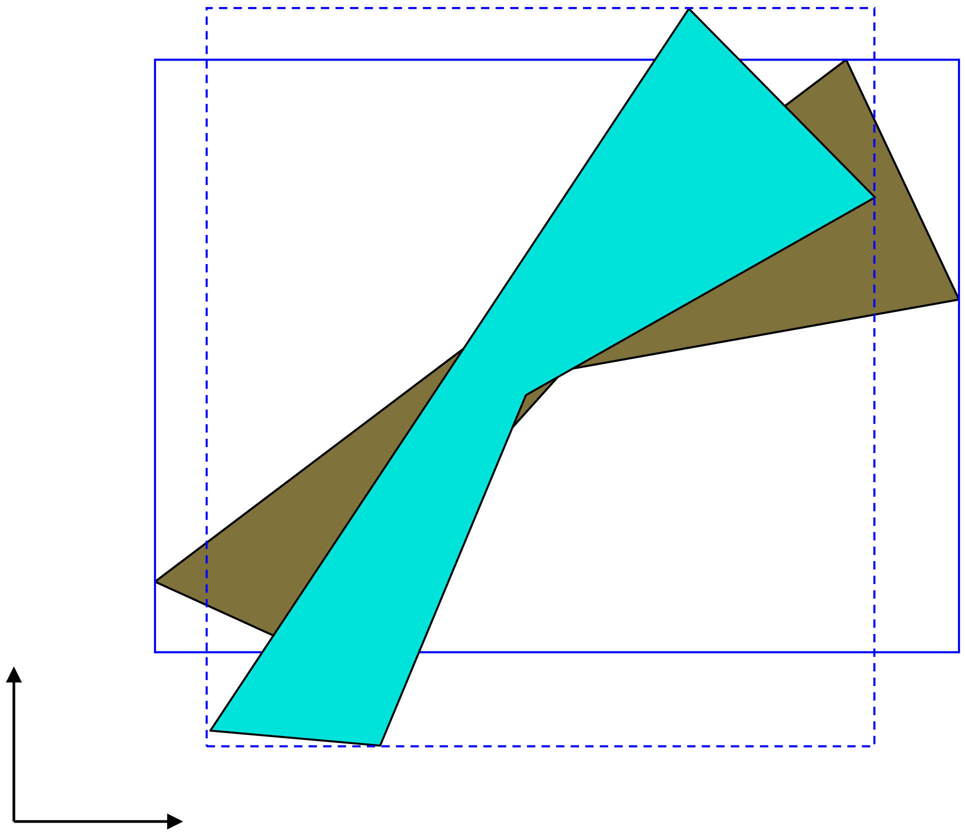
AABBs

- Axis-Aligned Bounding Boxes (AABBs)
  - Bound object with one or more boxes oriented along the same axis



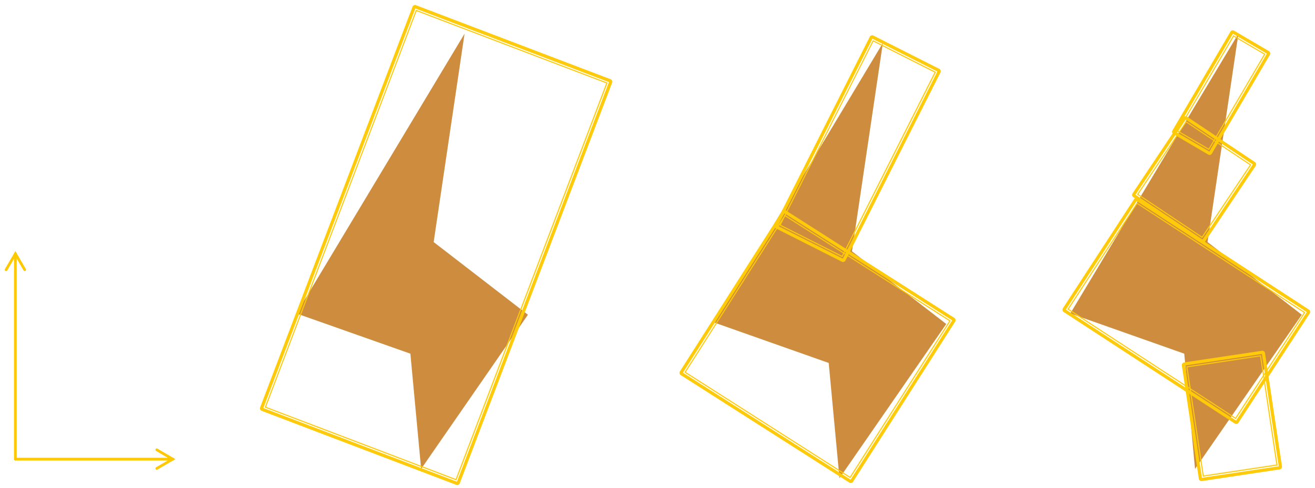
AABBs

- Not invariant
- Efficient
- Not tight



OBBs

- Oriented Bound Boxes (OBBs) are the same as AABBs except
  - The orientation of the box is not fixed



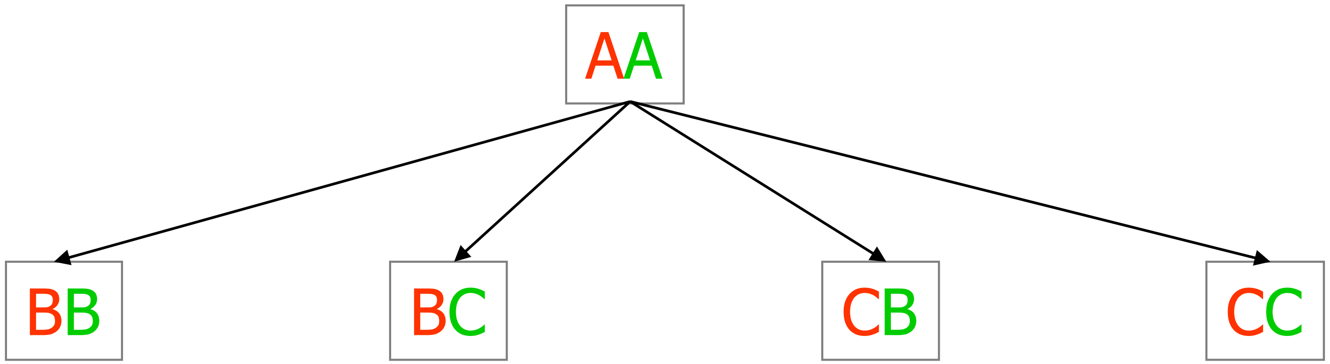
- OBBs can give you a tighter fit with fewer boxes

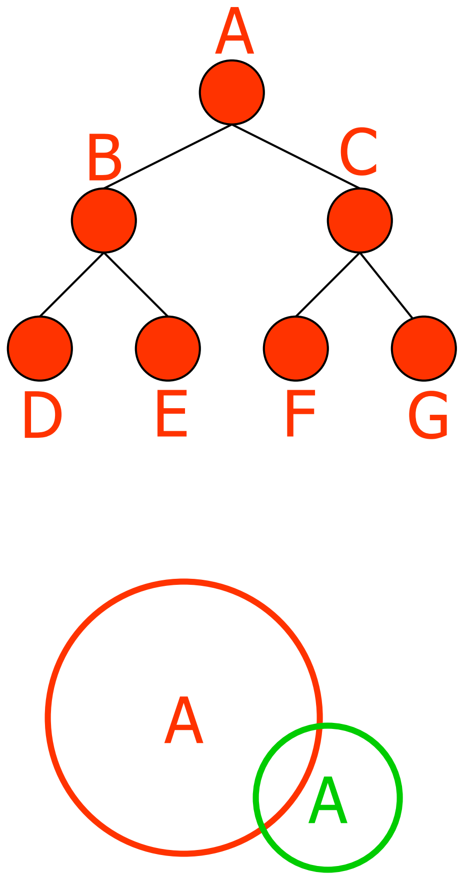
OBBs

- Invariant
- Less efficient to test
- Tight

Collision Detection using BVH

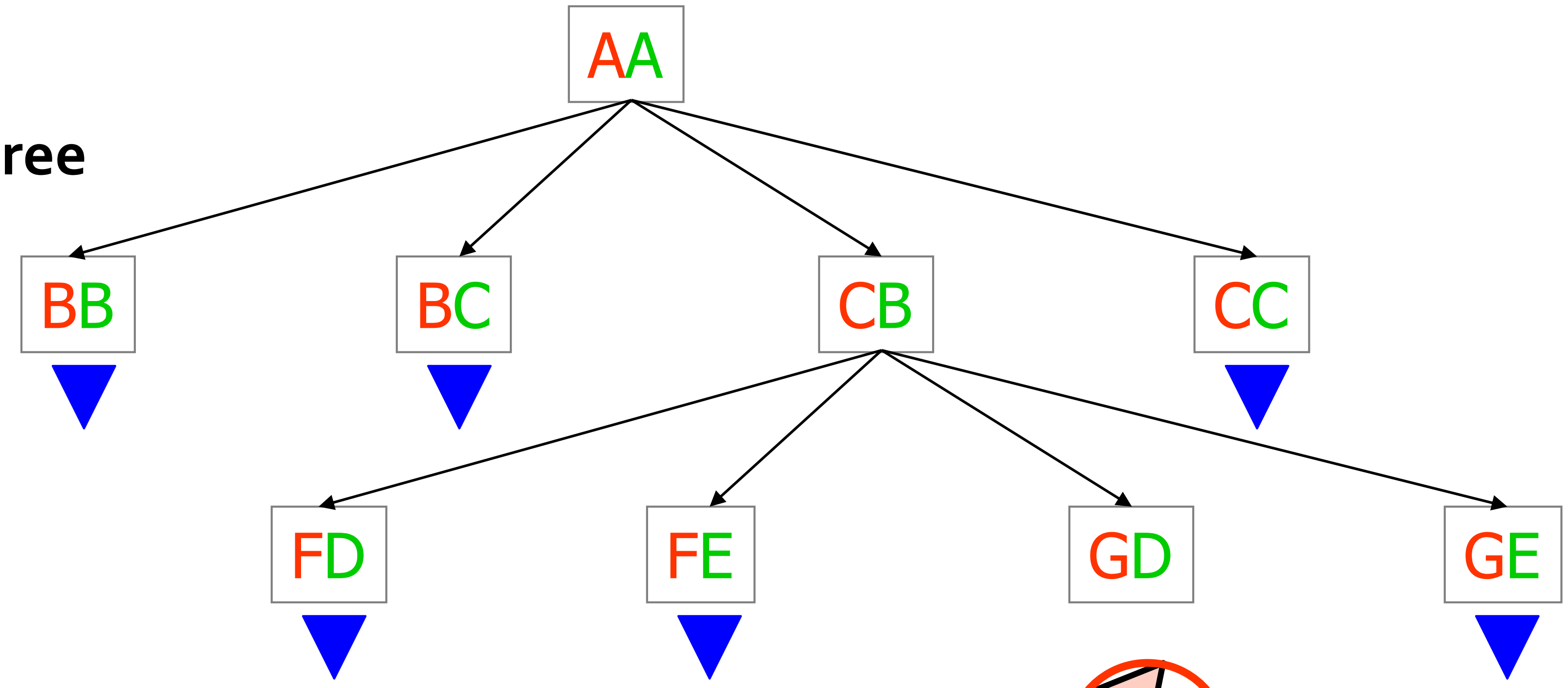
Search tree

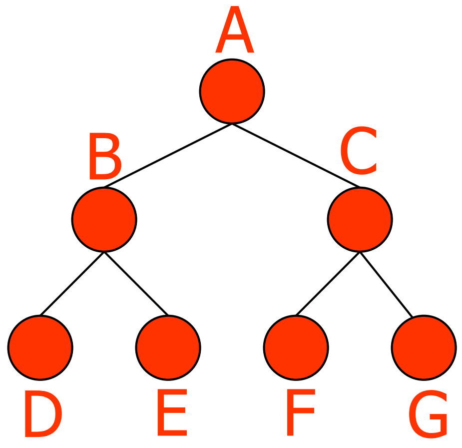




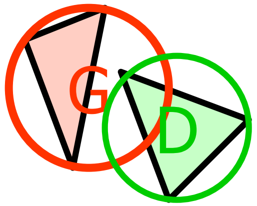
Collision Detection with BVH

Search tree



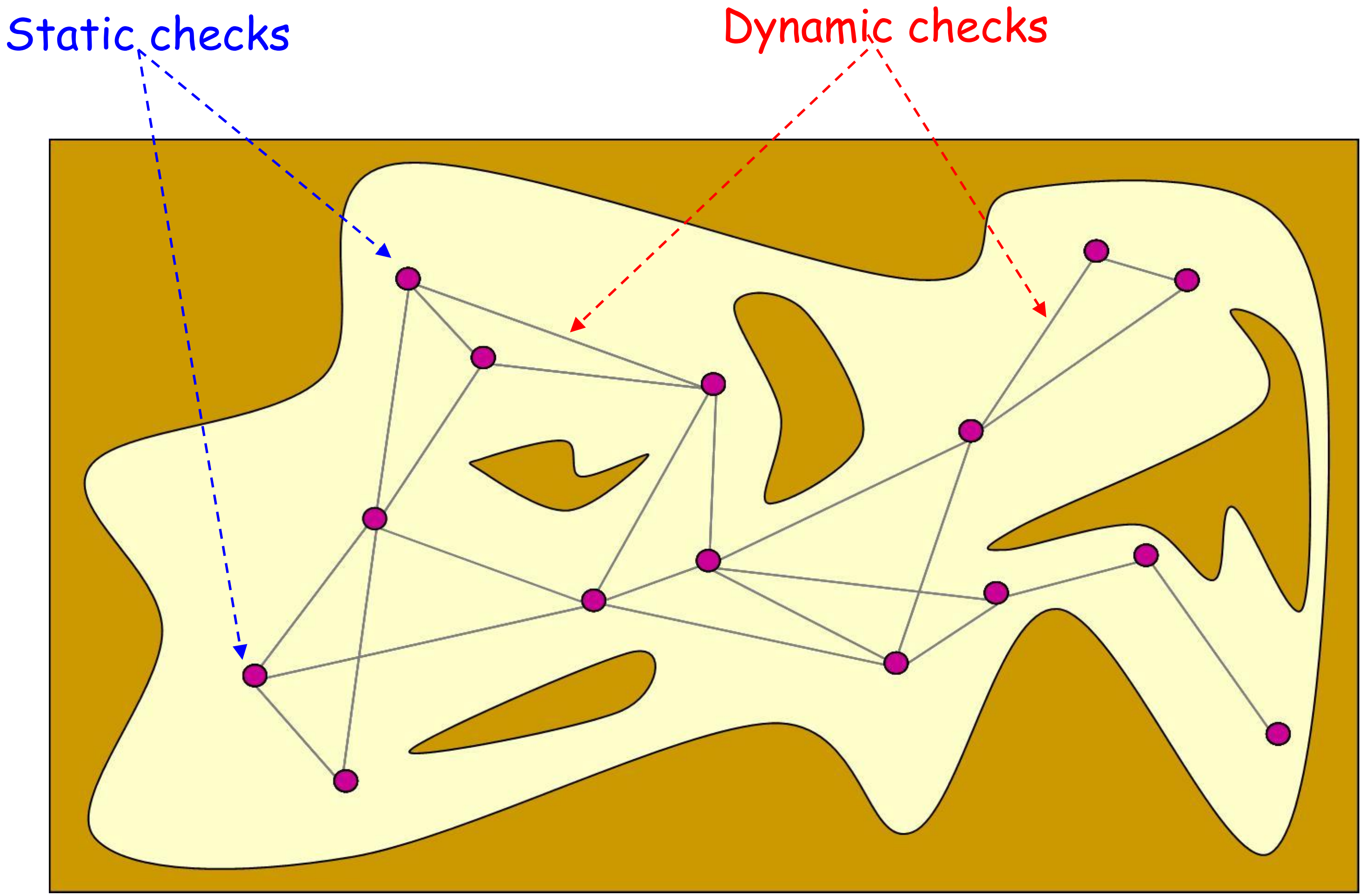


If two leaves of the BVH's overlap (here, **G** and **D**) check their content for collision



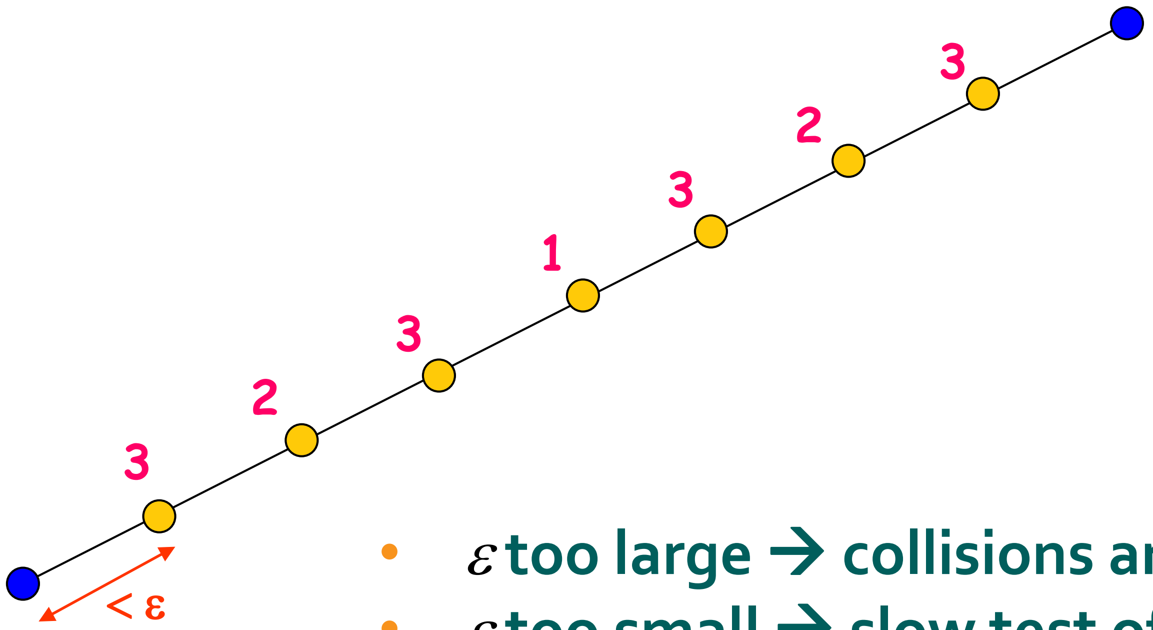
Dynamic collision checking

Static vs. Dynamic VS Collision Detection



Usual Approach to Dynamic Checking

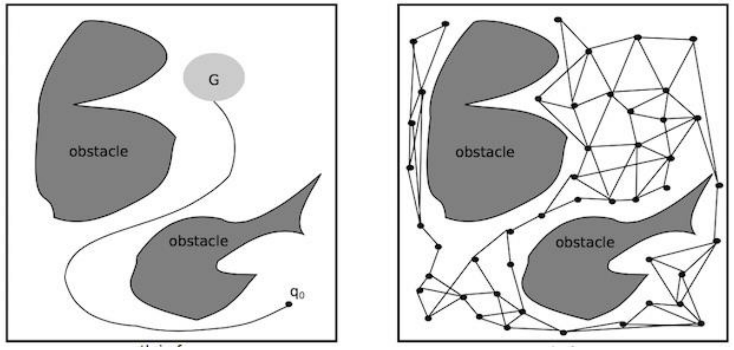
- Discretize path at some fine resolution ε
- Test statically each intermediate configuration



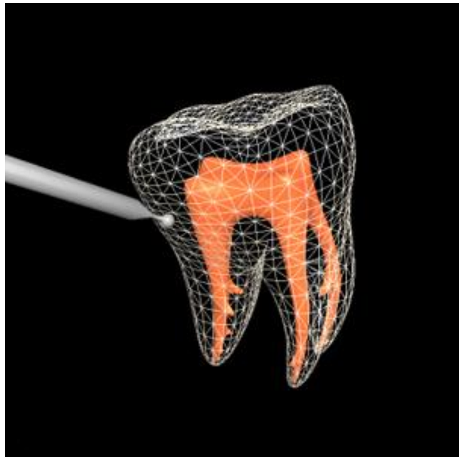
- ε too large → collisions are missed
- ε too small → slow test of local paths

Testing Path Segment vs. Finding First Collision

- PRM planning
  - Detect collision as quickly as possible → Bisection strategy



- Physical simulation, haptic interaction
  - Find first collision → Sequential strategy



Collision Checking for Moving Objects

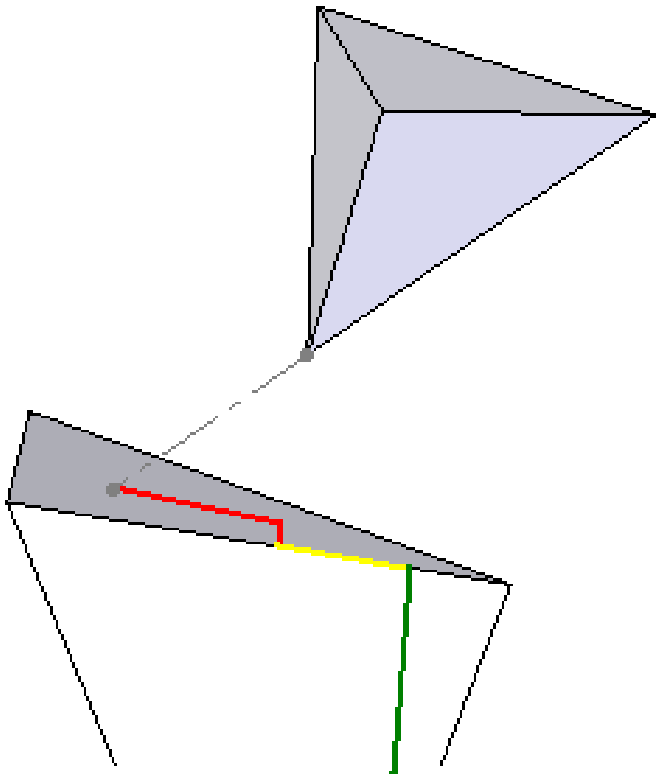
- Feature Tracking
- Swept-volume intersection

Feature tracking

- Compute the Euclidian distance of two polyhedra
  - Each object is represented as a **convex polyhedron** (or a set of polyhedra)
  - Each polyhedron has a field for its faces, edges, vertices, positions and orientations ← **features**
  - The closest pair of features between two polyhedra
    - The pair of features which contains the **closest points**
  - Given two polyhedra, find and keep updating their **closest features** (see [1])

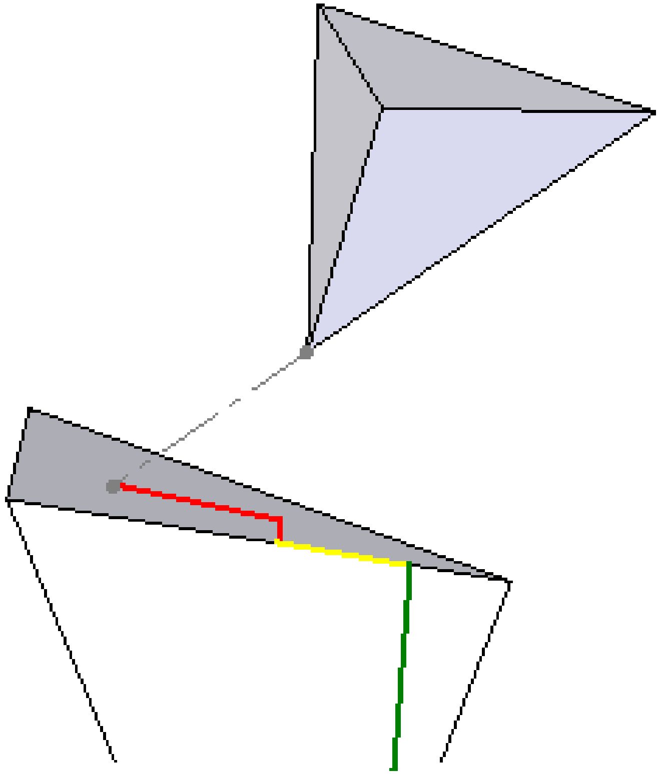
Feature Tracking

- Strategy
  - The **closest pair of features** (vertex, edge, face) between two polyhedral objects are computed **at the start configurations** of the objects
  - During motion, at each small increment of the motion, they are updated

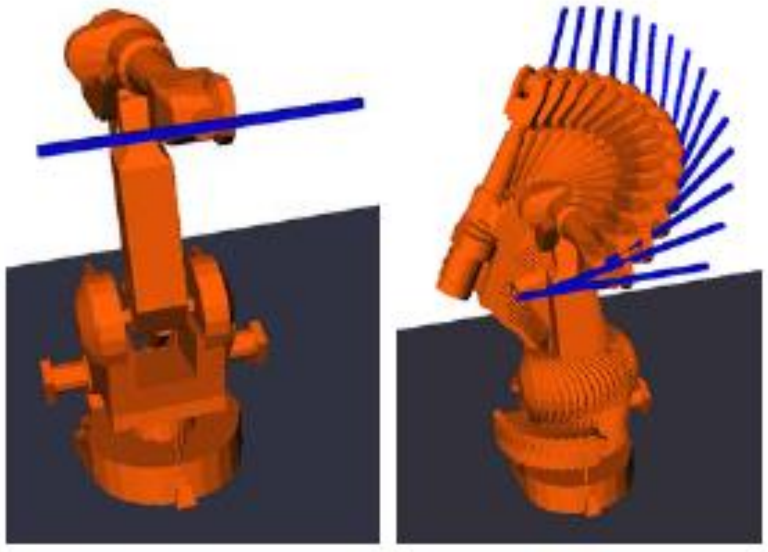


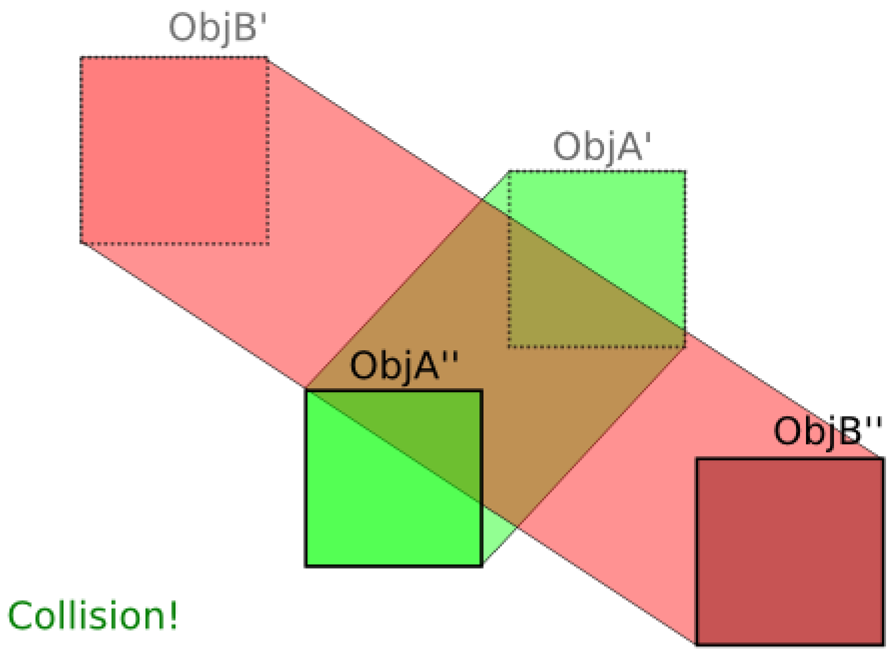
Feature Tracking

- Efficiency derives from two observations
  - The pair of closest features changes relatively **infrequently**
  - When it changes the new closest features will usually be on a **boundary** of the previous closest features



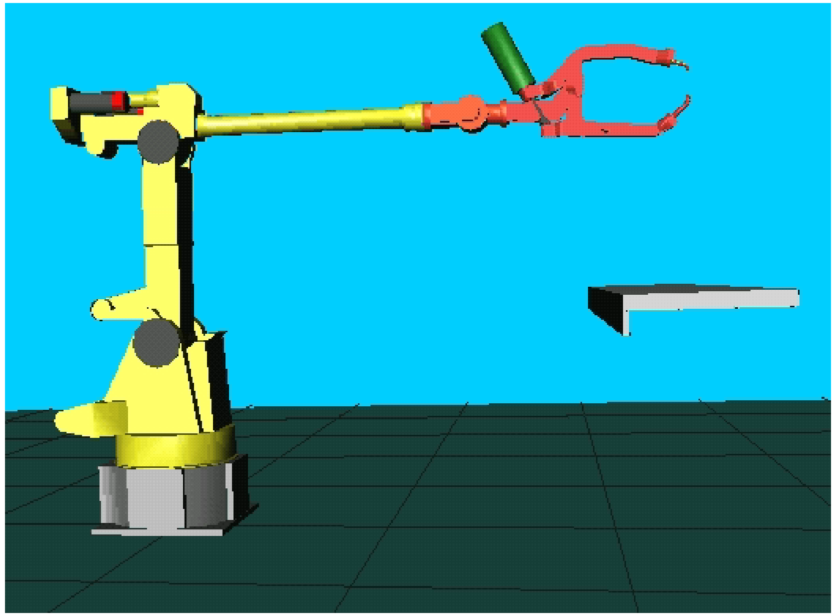
Swept-volume Intersection

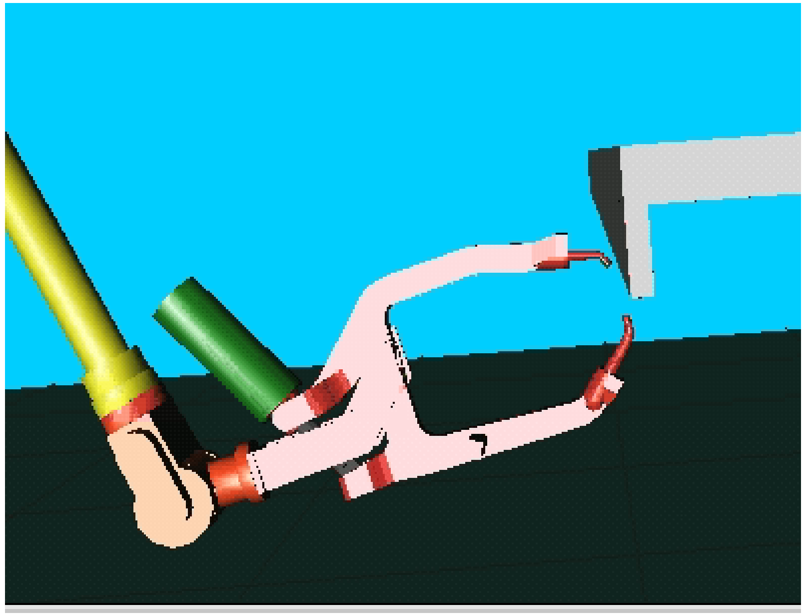




- ε too large → collisions are missed
- ε too small → slow test of local paths

Swept-volume Intersection





- ε too large → collisions are missed
- ε too small → slow test of local paths

Comparison

- Bounding-volume (BV) hierarchies
  - Discretization issue
- Feature-tracking methods
  - Geometric complexity issue with highly non-convex objects
- Swept-volume intersection
  - Swept-volumes are expensive to compute. Too much data.

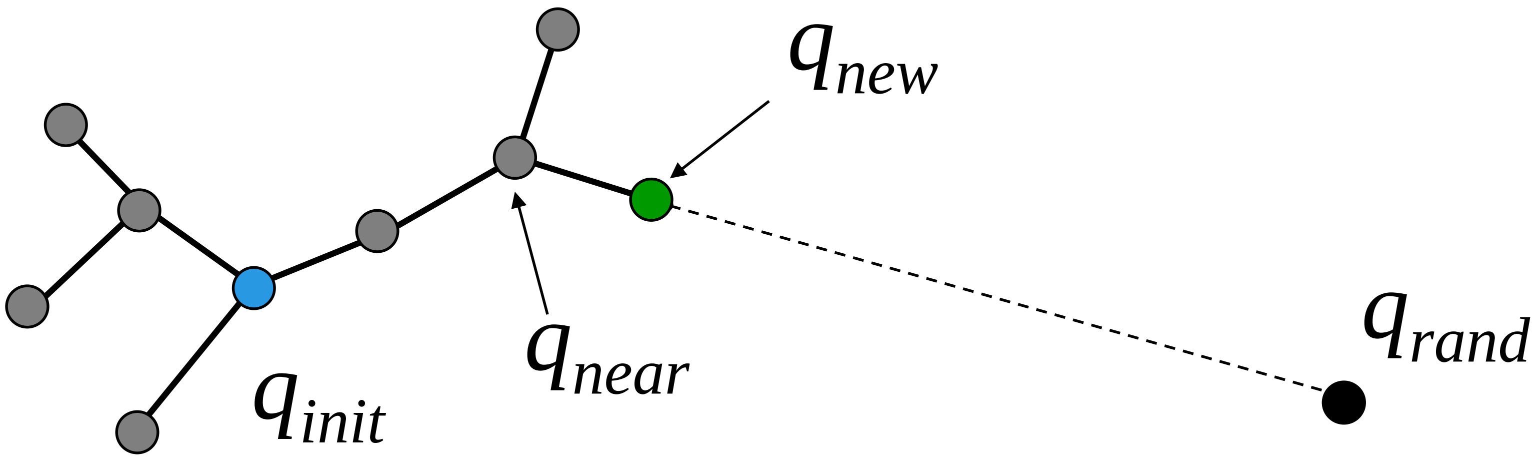
Reference

- [1] M. Lin and J. Canny. A Fast Algorithm for Incremental Distance Calculation. Proc. IEEE Int. Conf. on Robotics and Automation, 1991

Non-holonomic planning

RRT for non-holonomic planning problem

- We have learned about RRTs….



- But the standard version of sampling-based planners assume the robot can move in any direction at any time
- What about robots that **can't** do this?

Overview

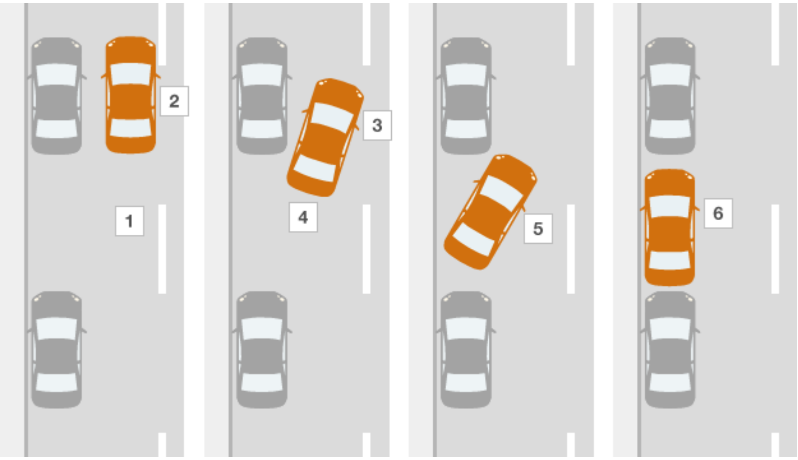
- Non-Holonomic constraints
  - Definition and examples
- Discrete Non-Holonomic Planning
- Sampling-based Non-Holonomic Planning

Holonomic vs. Non-Holonomic Constraints

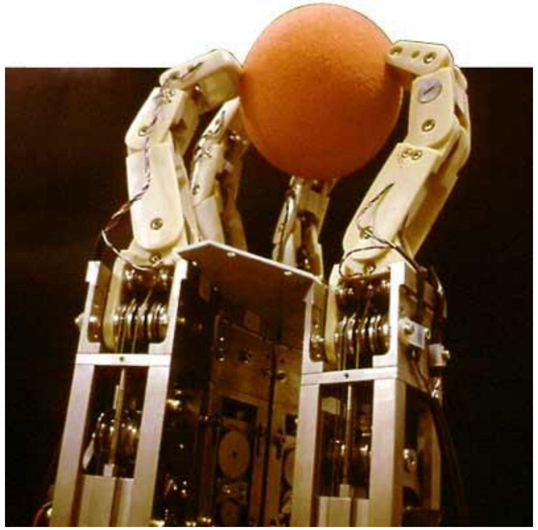
- **Holonomic** constraints depend only on **configuration**
  - $F(q, t) = 0$ (note they can be **time-varying**!)
  - Is collision constraint holonomic?

- **Non-holonomic** constraints are constraints that **cannot** be written in this form

Examples – Rolling without slipping

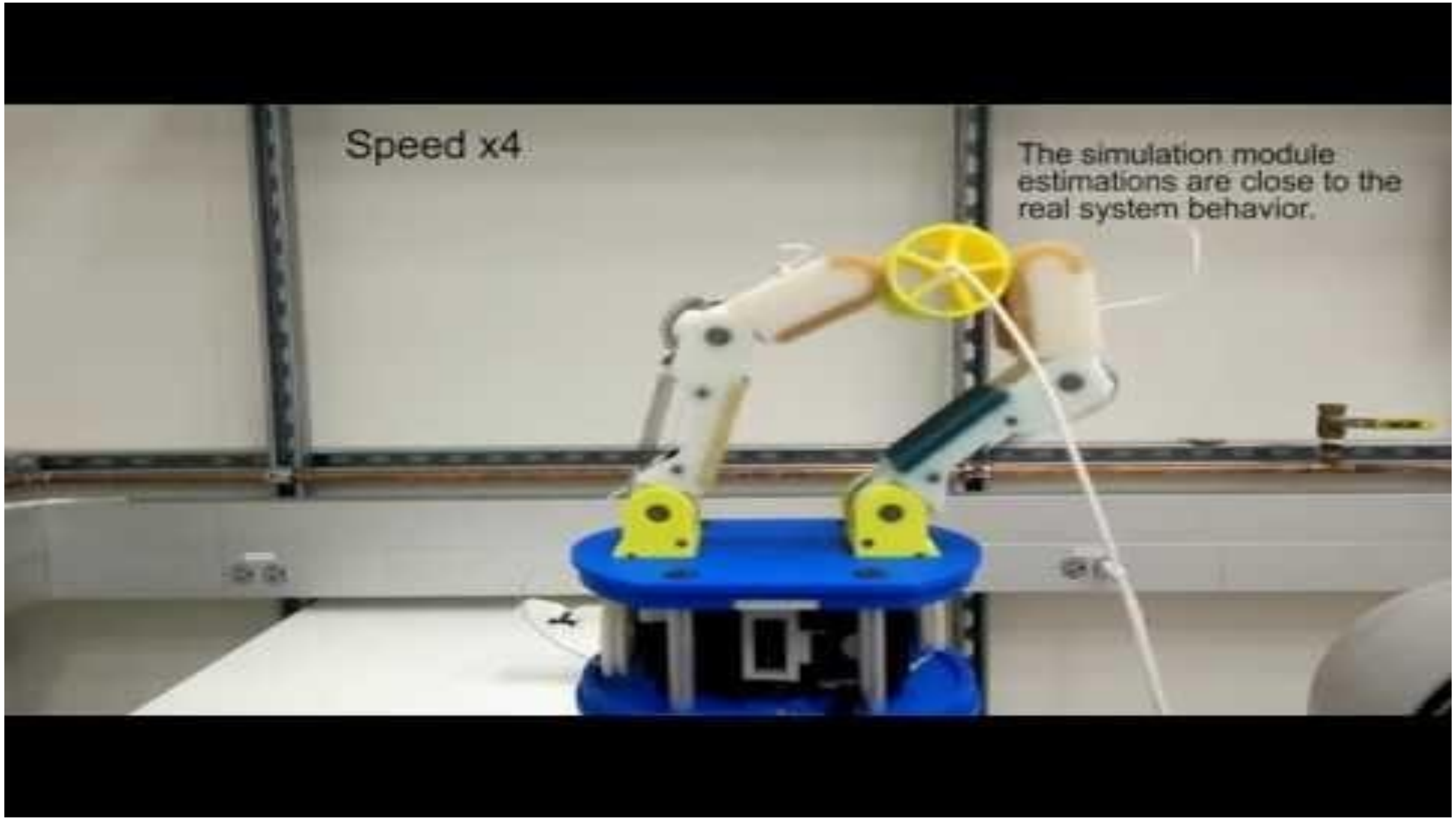


Parallel Parking

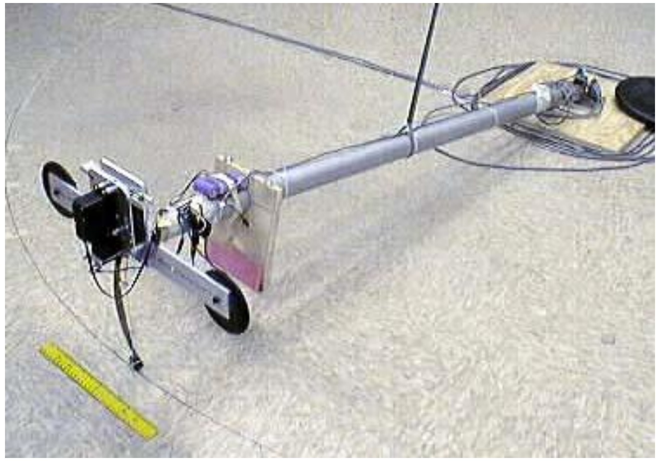


Manipulation with a robotic hand
Multi-fingered hand from Nagoya University

Dexterous, in-hand manipulation



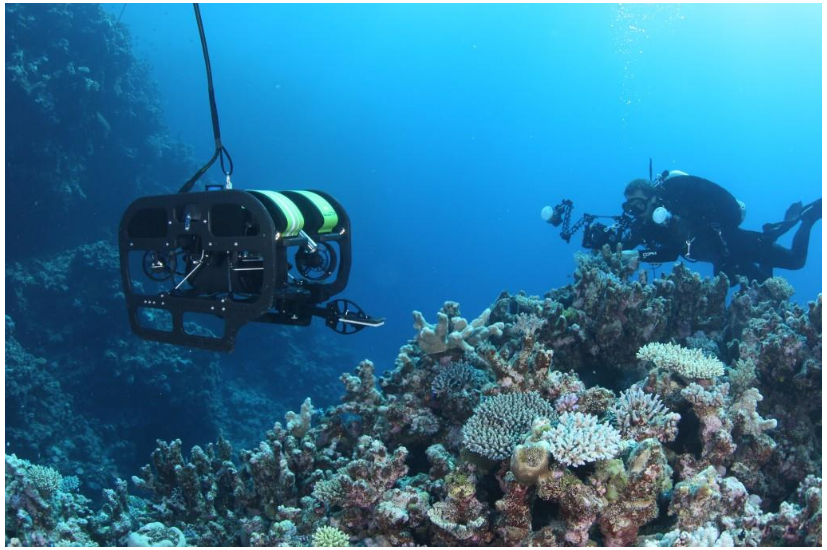
Example – Conservation of Angular Momentum



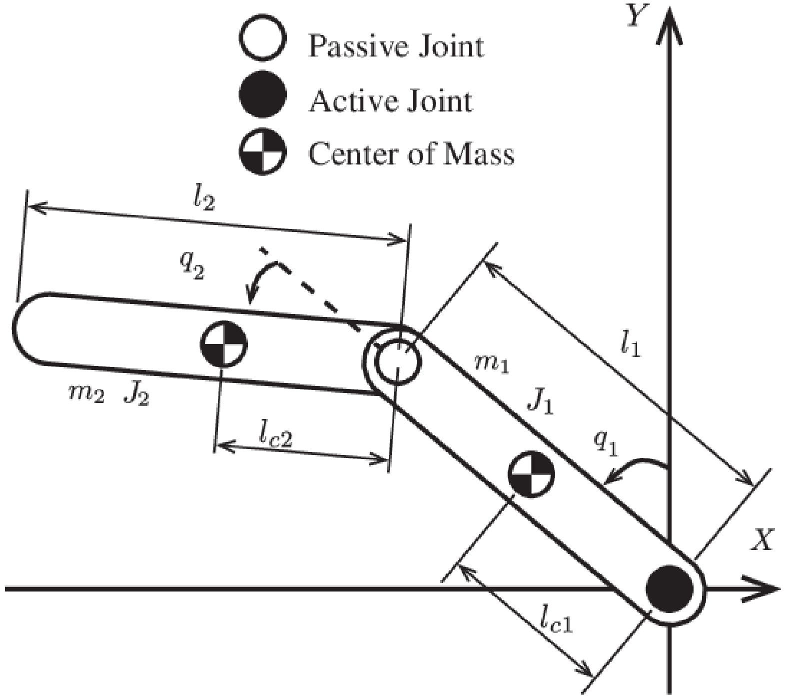
Hopping robots – RI's bow leg hopper (CMU)

AERcam, NASA - Untethered space robots

Example – Underactuation



Underwater robot
Forward propulsion is allowed
only in the pointing direction

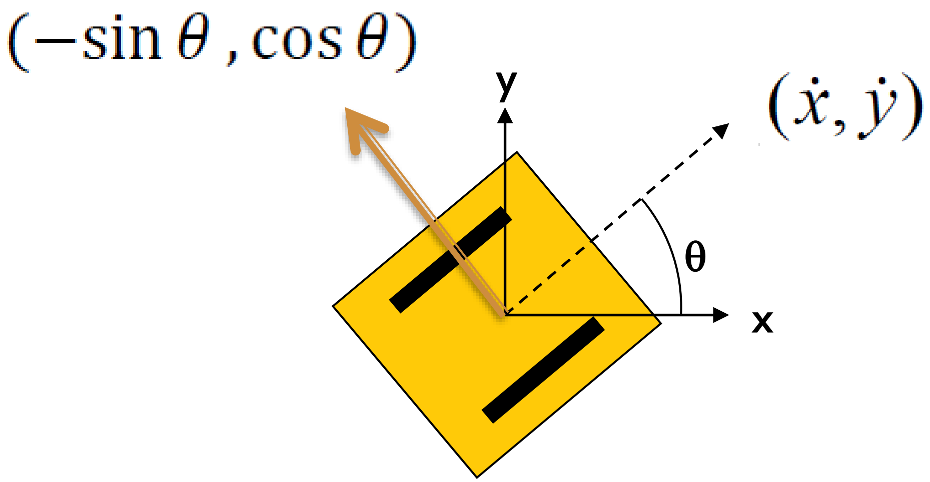


Robotic Manipulator
with passive joints

Mathematical Representation

- Constraint equation

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0$$



- What does this equation tell us?
  - The direction we can't move in
    - If $\theta=0$, then the velocity in y = 0
    - If $\theta=90$, then the velocity in x = 0
  - Write the constraint in matrix form

Mathematical Representation

- Write the constraint in matrix form

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad \dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

Position & Velocity Vectors

$$w_1(q) \cdot \dot{q} = 0 = [-\sin\theta \ \ \cos\theta \ \ 0] \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \Rightarrow -\dot{x}\sin\theta + \dot{y}\cos\theta = 0$$

$$w_1(q) = [-\sin\theta \ \ \cos\theta \ \ 0]$$

Constraint Vector

Holonomic vs. Non-Holonomic Constraints

- Example: The kinematics of a unicycle
  - Can move forward and back
  - Can rotate about the wheel center
  - Can't move sideways

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0$$

Holonomic vs. Non-Holonomic Constraints

- Can we just integrate them to get a holonomic constraint?
  - Intermediate values of its trajectory matters

- Can we still reach any configuration (x,y,θ)?
  - No constraint on configuration, but …
  - May not be able to go to a (x,y,θ) **directly**

Holonomic vs. Non-Holonomic Constraints

- Non-holonomic constraints are **non-integrable**
  - Thus non-holonomic constraints must contain **derivatives of configuration**

- In this case, how to move between configurations (or states) when planning?
  - E.g., in RRT, we assumed we can move between arbitrary nearby configurations using a straight line. But now …

Example – A simple car

- State Space

$$x, y, z, \psi, \varphi, \theta$$
$$\dot{x}, \dot{y}, \dot{z}, \dot{\psi}, \dot{\varphi}, \dot{\theta}$$

- Control space
  - Speed or Acceleration
  - Steering angle

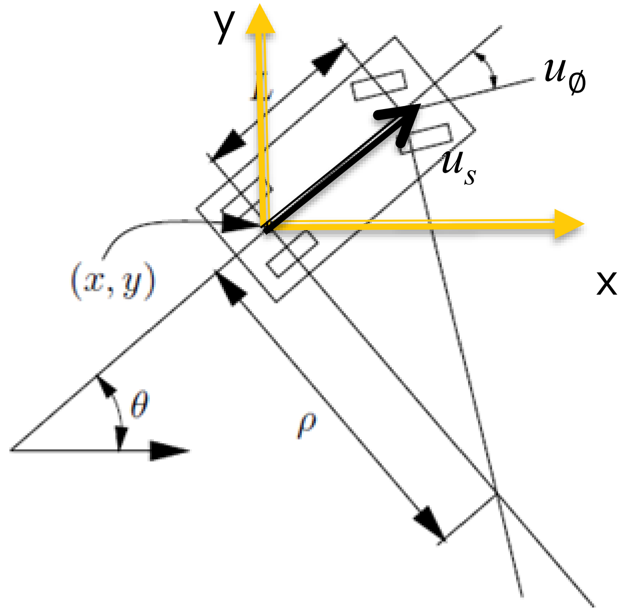
Example – A simple car

- Non-holonomic Constraint
  - In a small time interval, the car must move approximately in the **direction that the rear wheels are pointing**.

- Motion model
  - u_s = speed
  - u_ϕ = steering angle

$$\Delta T \to 0, \quad \frac{dx}{dy} = \frac{\dot{x}}{\dot{y}} = \tan\theta$$

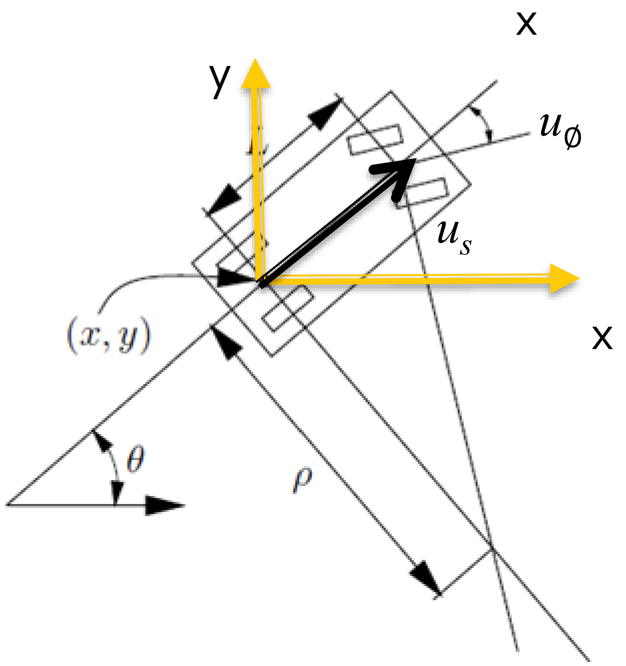
$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0$$



Example – A simple car

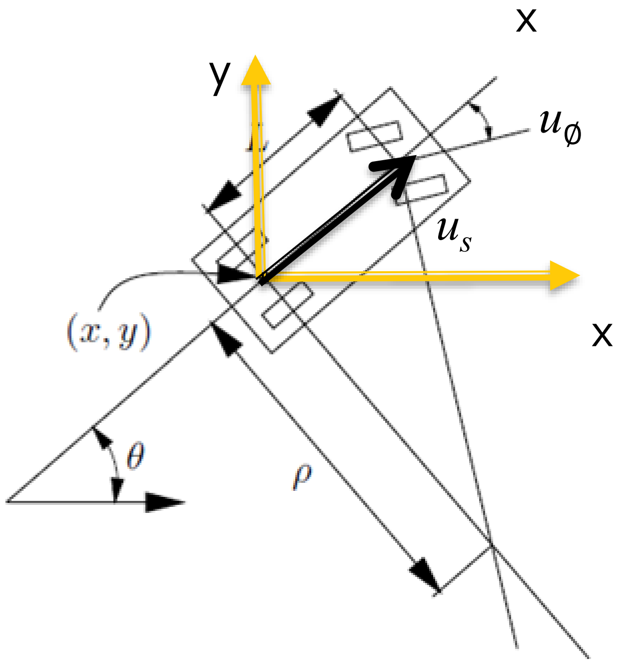
- Motion model
  - u_s = speed

$$\dot{x} = u_s \cos\theta, \quad \dot{y} = u_s \sin\theta$$



Example – A simple car

- Motion model
  - u_ϕ = steering angle
    - If the steering angle is fixed, the car travels in a circular motion → radius ρ
    - Let ω denote the distance traveled by the car



Rotation of the vehicle

$$d\omega = \rho d\theta \qquad \frac{L}{\rho} = \tan u_\phi$$

$$\Rightarrow \quad d\theta = \frac{\tan u_\phi}{L} d\omega \qquad \dot{\omega} = u_s$$

$$\Rightarrow \quad \boxed{\dot{\theta} = \frac{u_s}{L} \tan u_\phi}$$

Example – A simple car

- We have derived the model

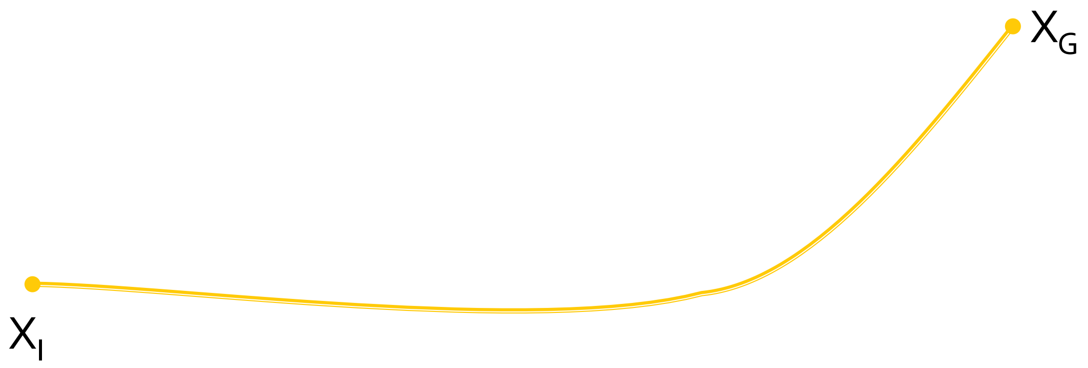
$$\dot{x} = u_s \cos\theta, \quad \dot{y} = u_s \sin\theta$$

$$\dot{\theta} = \frac{u_s}{L} \tan u_\phi$$

- Now how to plan the trajectory given the start and end states of the mobile robot?

Moving Between States (with No Obstacles)

- Two-Point Boundary Value Problem (BVP):
  - Find a control sequence to take system from state X_I to state X_G while obeying kinematic constraints.



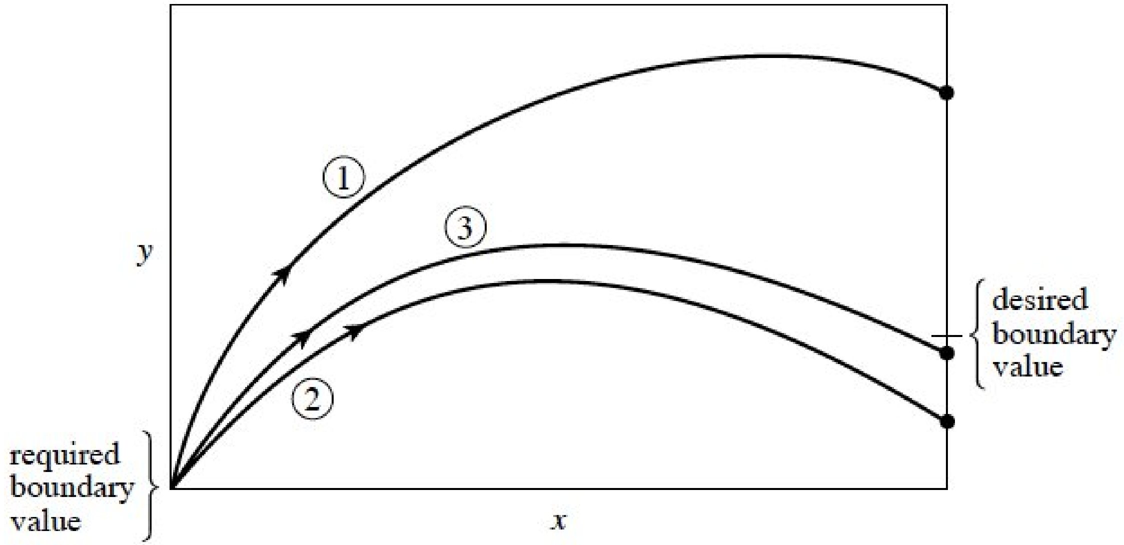
Shooting Method

- "Shoot" out trajectories in different directions until a trajectory of the desired boundary value is found.
  - System

$$\frac{dy}{dx} + f(x, y) = 0$$

  - Boundary condition

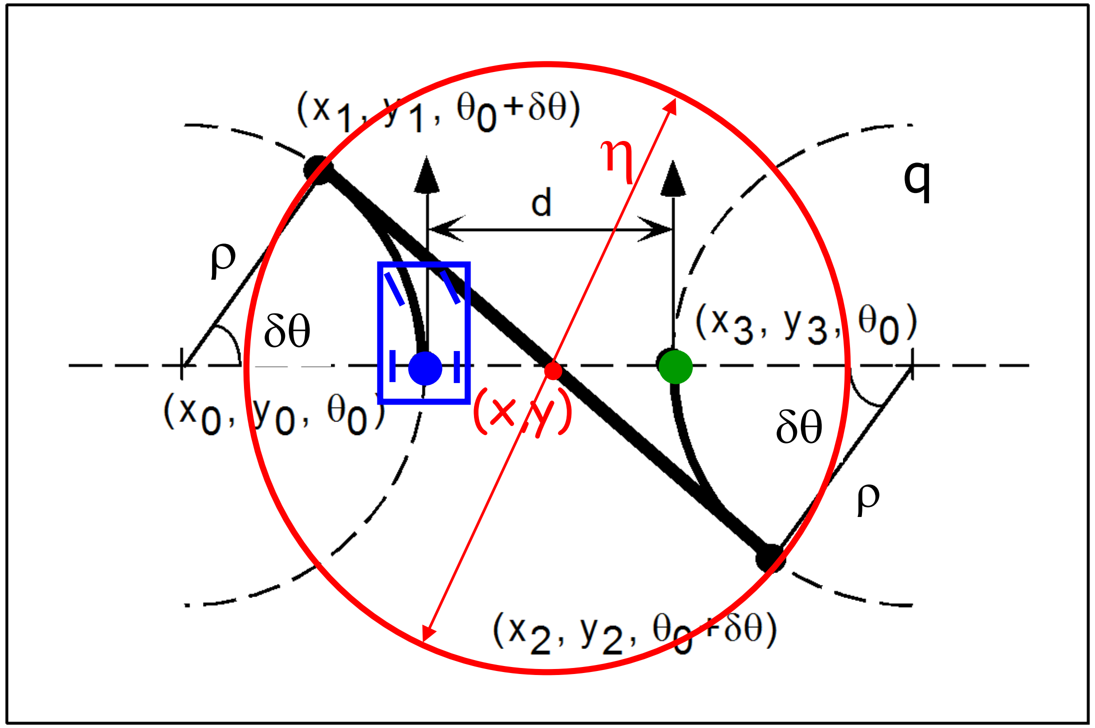
$$y(0) = 0,\ y(1) = 1$$



Alternative Method

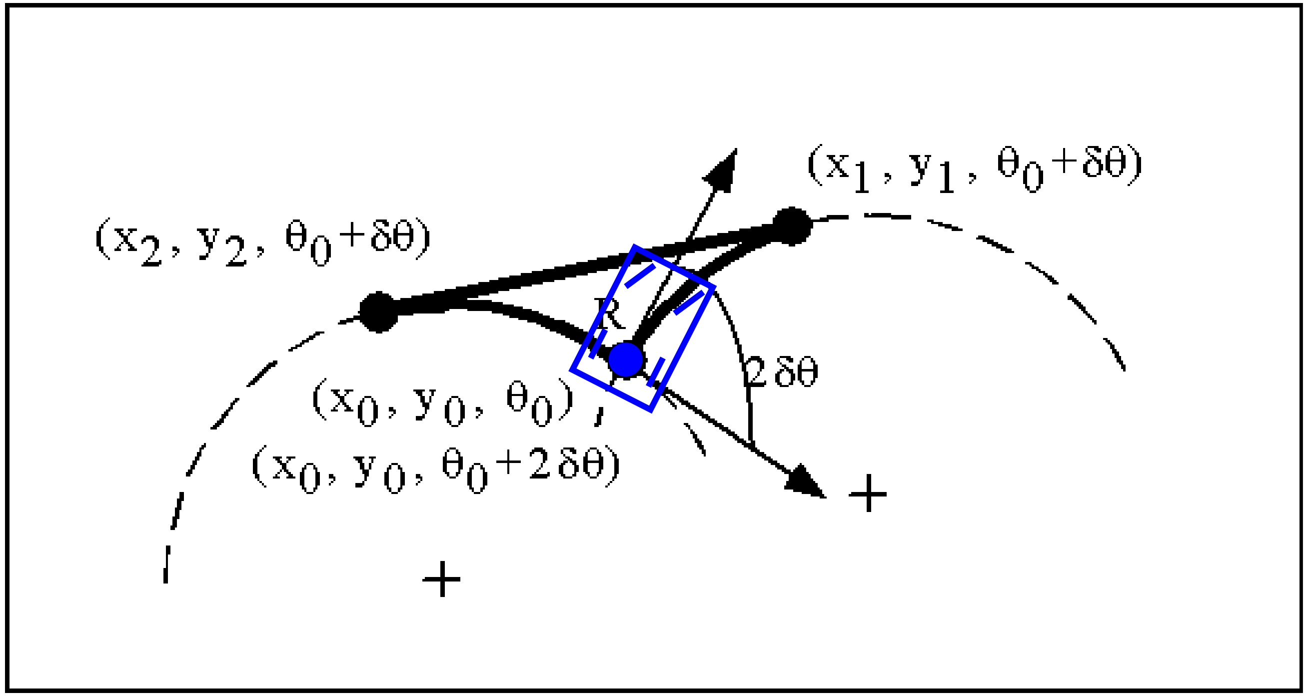
- Composites of maneuver primitives
  - Due to non-holonomic constraint, direct sideway motion is prohibited
  - Approximate the side way using a series of forward/backward and turning maneuvers

Type 1 Maneuver



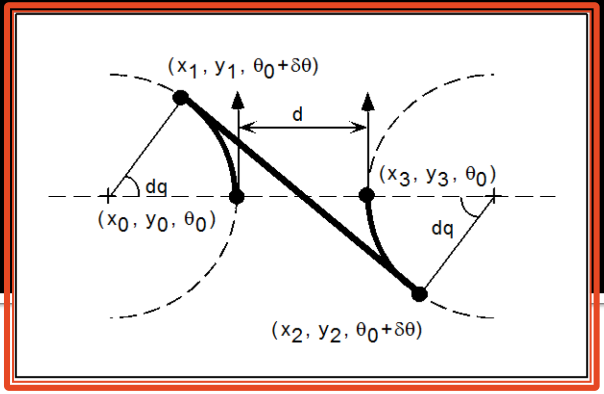
→ **Allows sidewise motion**

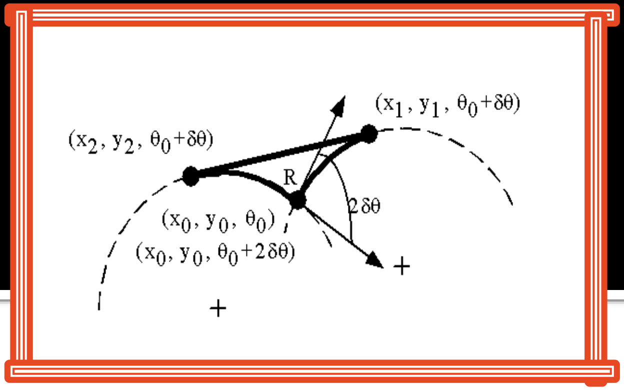
Type 2 Maneuver

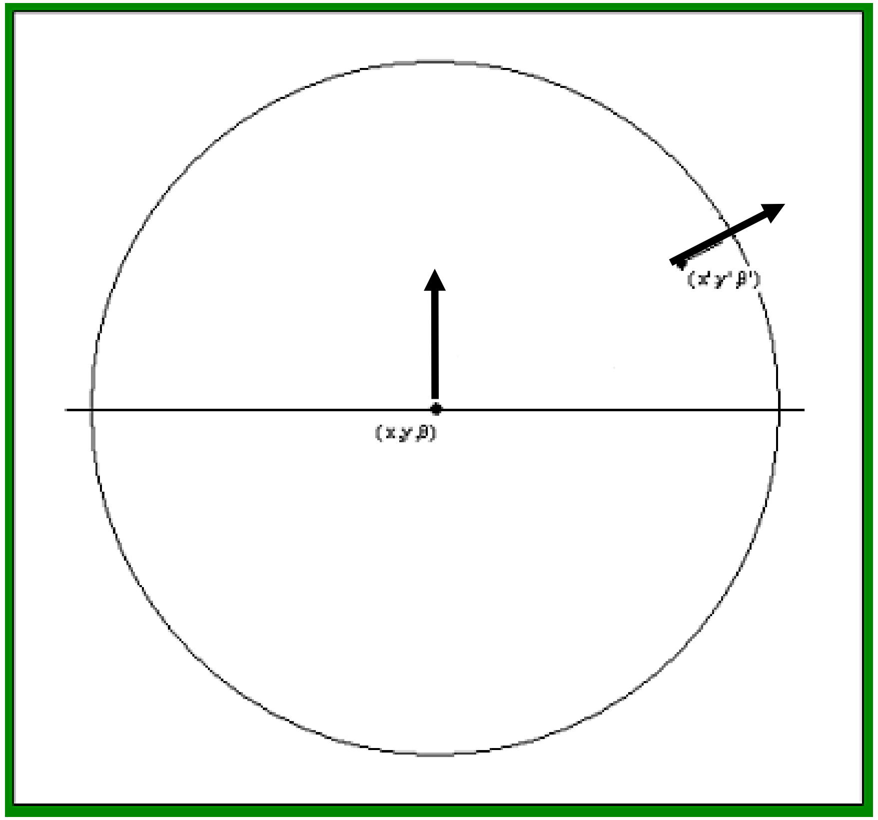


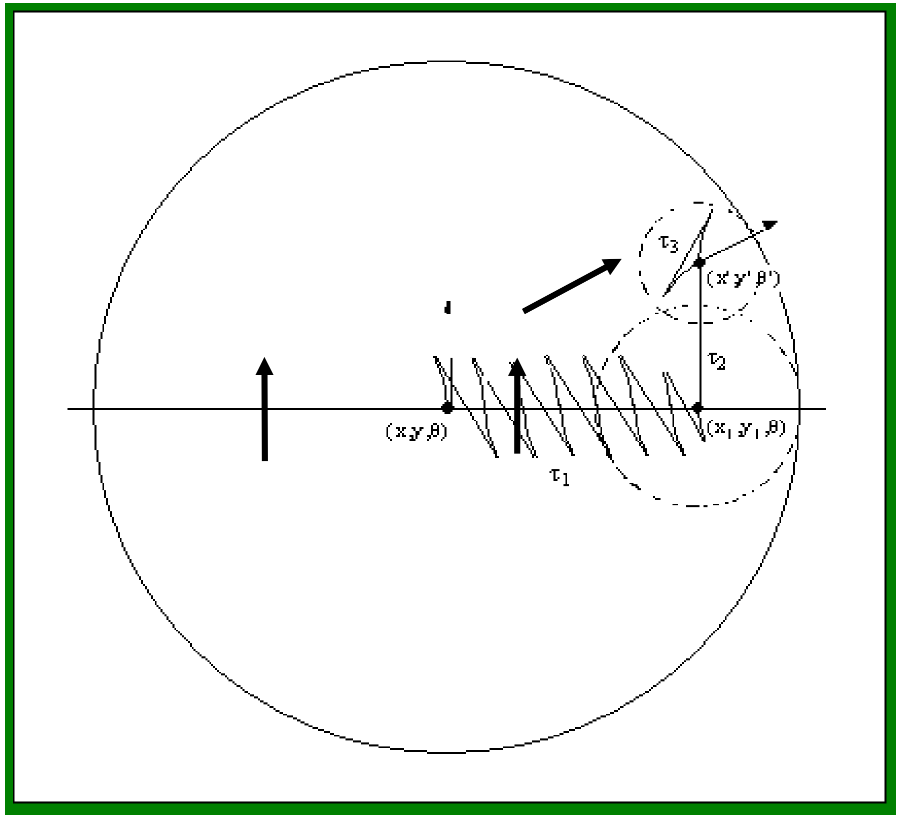
→ Allows pure rotation

Combination

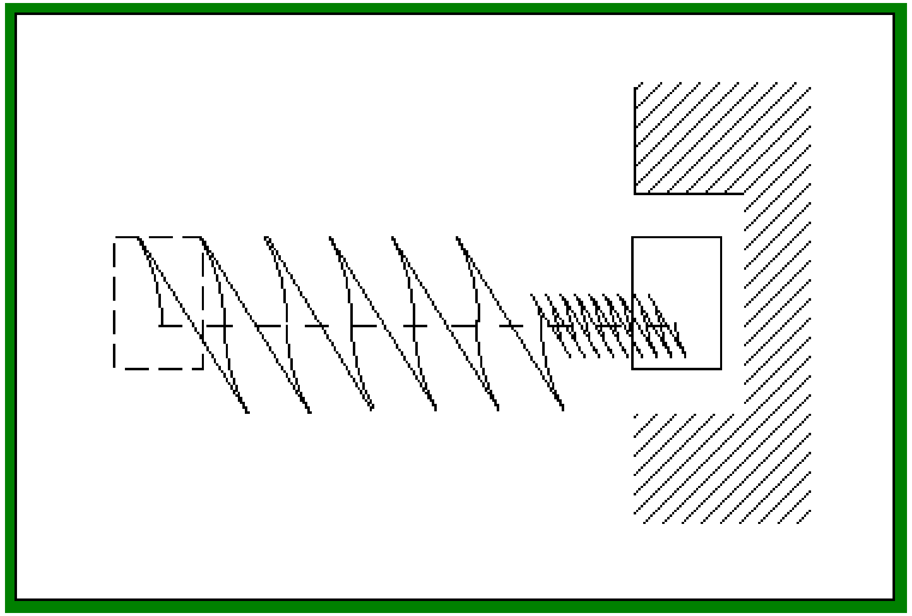


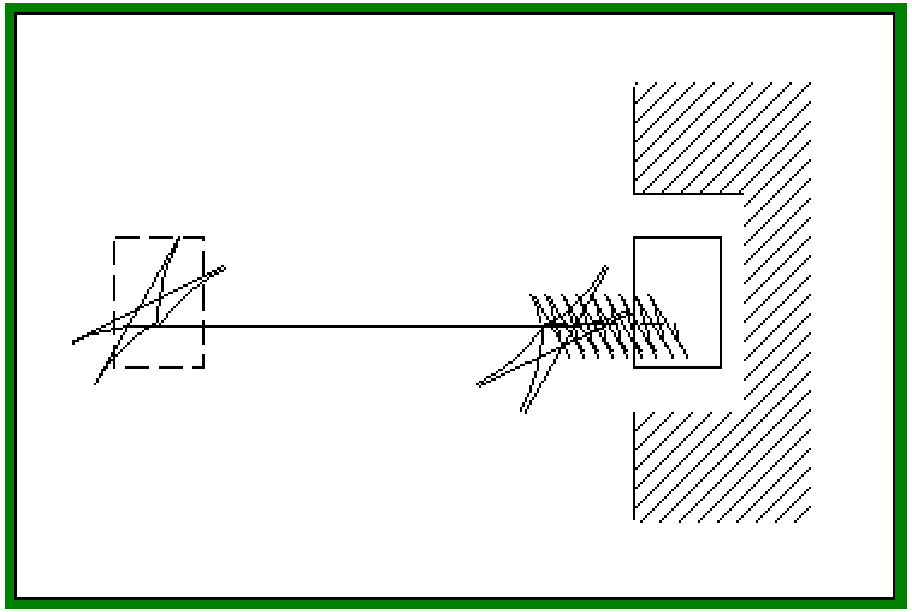






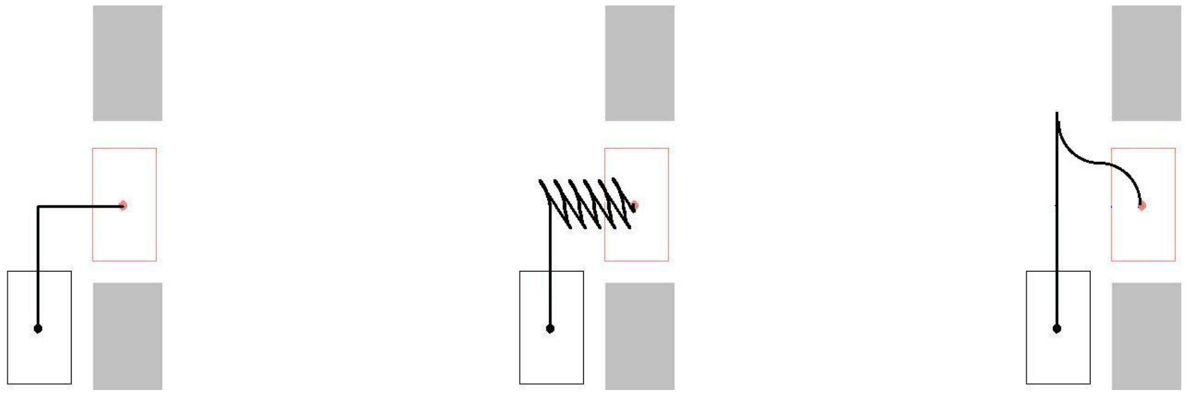
Path Examples





Drawbacks

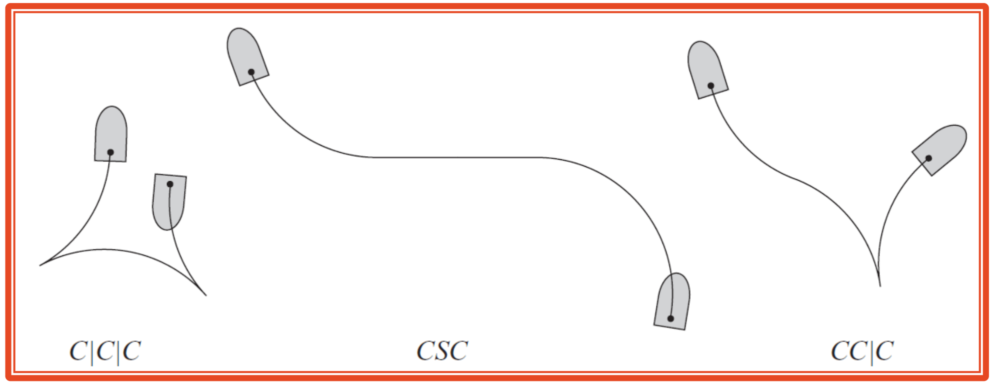
- Final path can be far from optimal



- Not applicable to car that can only move forward
  - e.g., an airplane

Optimal Solution?

- Reed and Shepp (RS) Path
  - Optimal path must be **a discrete and computable set of curves**
  - Each member of this set consists of sequential straight-line segments and circular arcs at the car's **minimum turning radius**
- Notation
  - C – curve
  - S – straight line
  - "|" – switch direction
  - Subscript – traverse distance



Reeds and Shepp Paths

- Given any two configurations
  - The shortest RS paths between them is also the **Optimal** path
  - The optimal path is guaranteed to be contained in the following set of path types

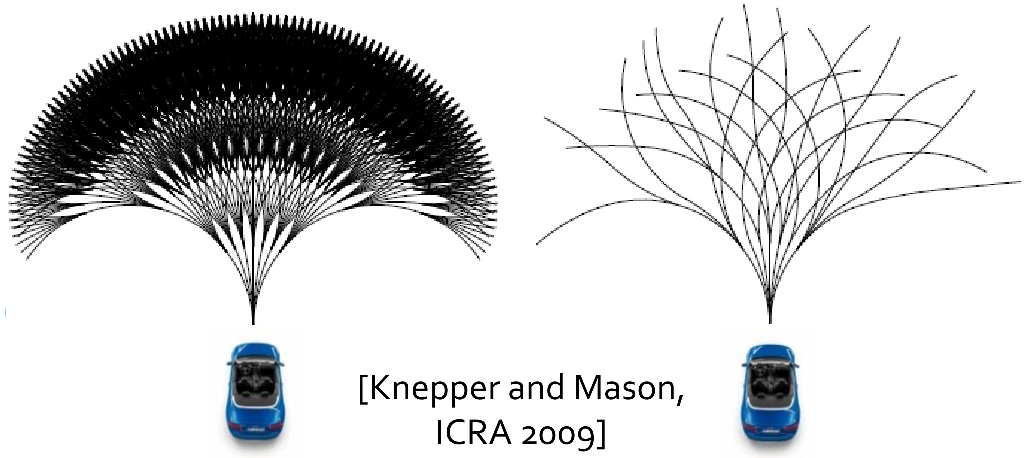
$$\{C|C|C,\ \ CC|C,\ \ C|CC,\ \ CC_a|C_aC,\ \ C|C_aC_a|C,$$
$$C|C_{\pi/2}SC,\ \ CSC_{\pi/2}|C,\ \ C|C_{\pi/2}SC_{\pi/2}|C,\ \ CSC\}$$

- Strategy
  - Pre-compute a map indexed by the goal relative to the start configuration
  - Look up in the map for the optimal path – may not be unique

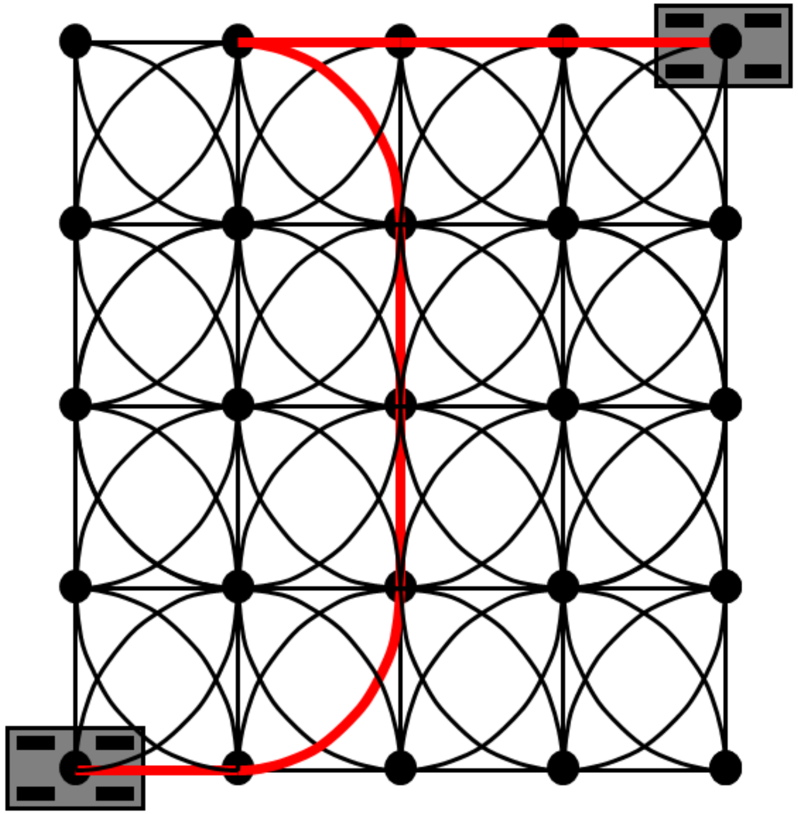
Example of Generated Path

Discrete Planning

- **Sequence of driving motion primitives**
  - Compute **State Lattice**
  - Search for a sequence of states



[Knepper and Mason, ICRA 2009]



Sequencing of Primitives

- Discretize control space [Barraquand & Latombe, 1993]
  - Discontinuous curvature
  - Cost = number of reversals
  - Dijkstra's Algorithm

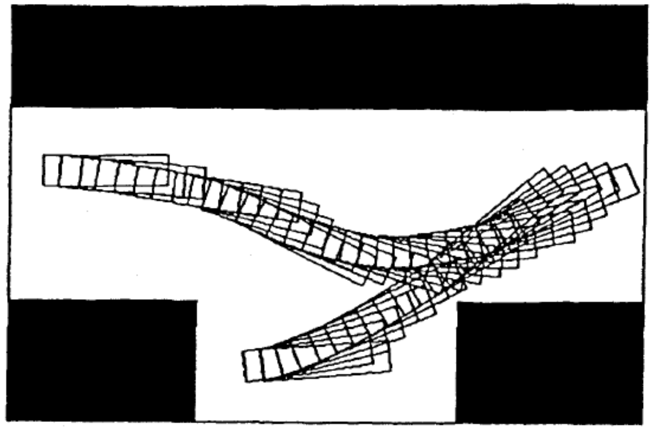


Fig. 4. Parking a car.

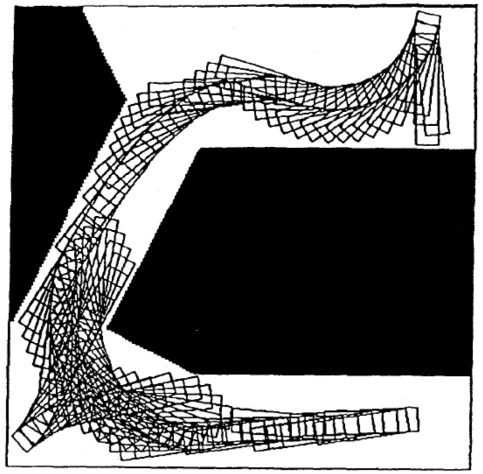
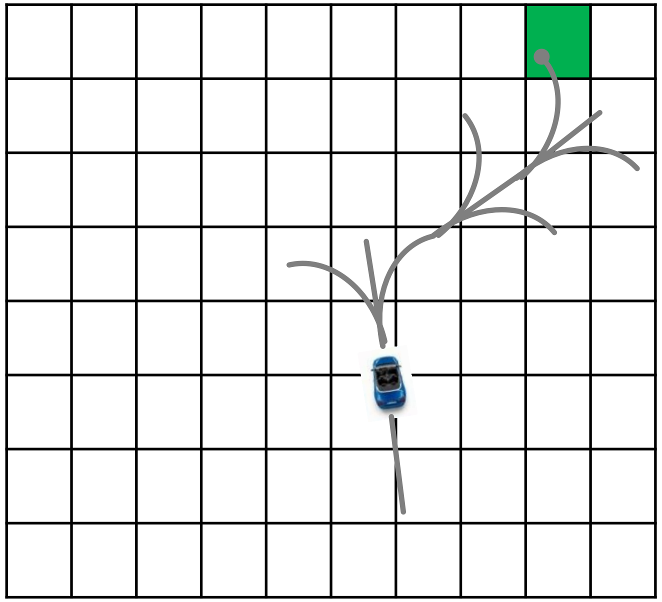
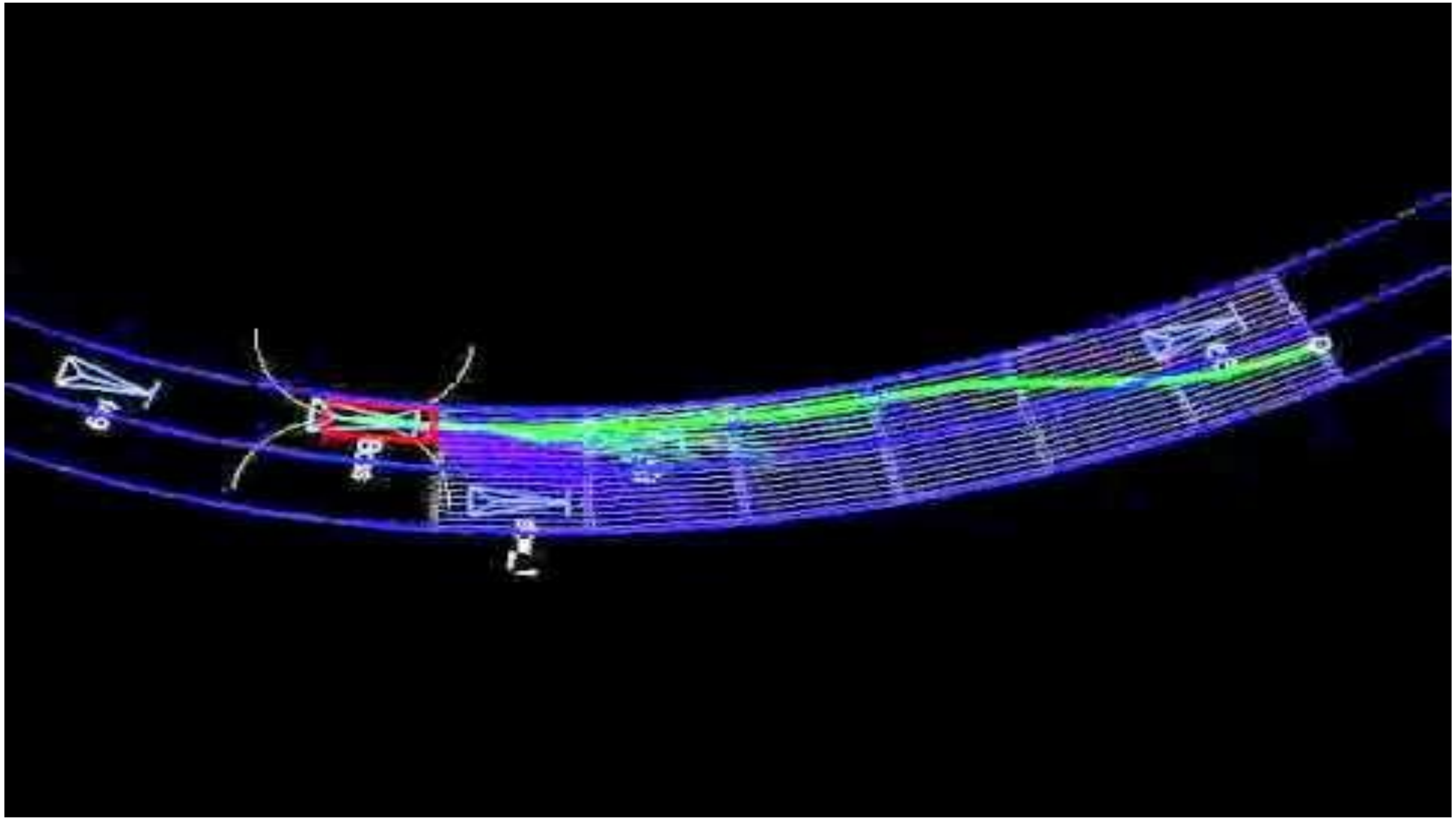


Fig. 5. Car maneuvering in a cluttered workspace.

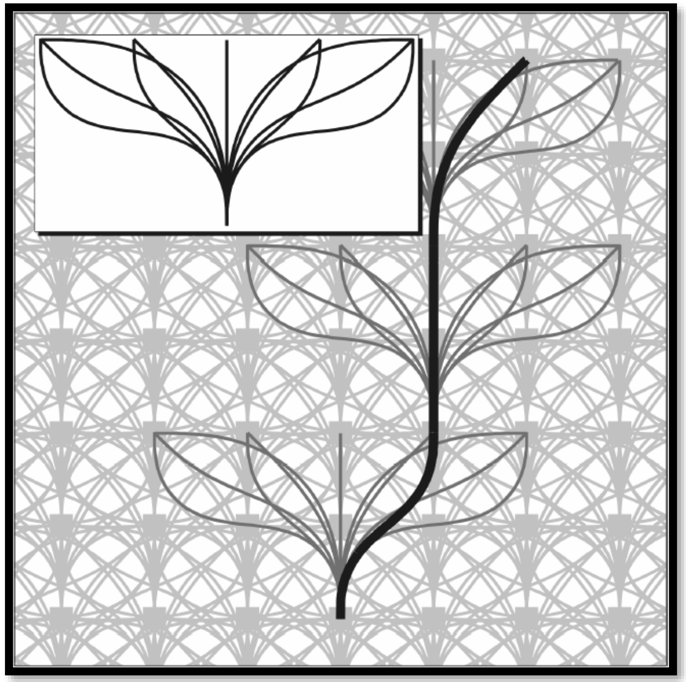


Search a path for merging between cars



State Lattice

- Two methods to get lattice
  - **Forward** – For certain systems, can sequence primitives to make lattice
  - **Inverse** – Discretize space, use BVP solvers to find trajectories between states

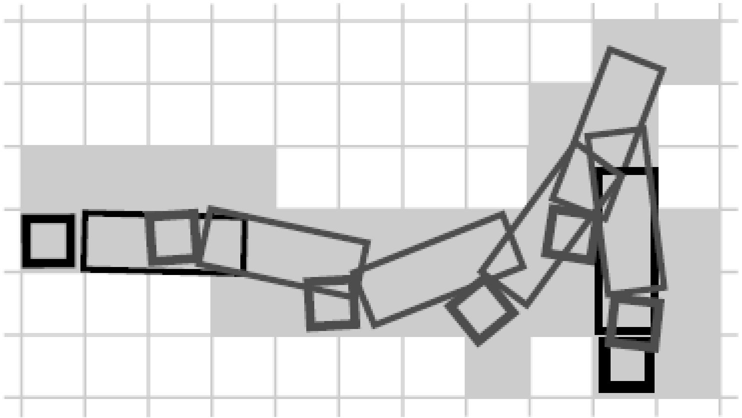


Sequencing of Primitives

- Choice of set of primitives affects
  - Completeness
  - Optimality
  - Speed

State Lattice

- Impose continuity constraints at graph vertices
- Search state lattice like any graph (i.e. A*)
- Pre-compute swept volume of robot for each primitive for faster collision checking



Pivtoraiko et al. 2009

Sampling-Based Planning

- Forming a full state lattice is **impractical** for high dimensions
  - So, sample instead.

- IMPORTANT
  - We are now sampling **state space** (position and velocity), not C-space (position only)

Challenges

- Curse of Dimensionality
  - **Dimension** of the space is **doubled** – position and velocity
- Local planner
  - Moving between points is harder (**can't go in a straight line**)
- **Distance metric** is unclear
  - Euclidian distance is not the correct metrics

PRM-style Non-Holonomic Planning

- Same as regular PRM
  - Sampling, graph building, and query strategies
- Problem
  - Local planner needs to reach an **EXACT** state (i.e. a given node) while obeying non-holonomic constraints

PRM-style Non-Holonomic Planning

- In general – BVP problem
  - use general solver (slow)
- In practice
  - Local planner specialized to system type
- Example
  - For Reeds-Shepp car, can compute optimal path

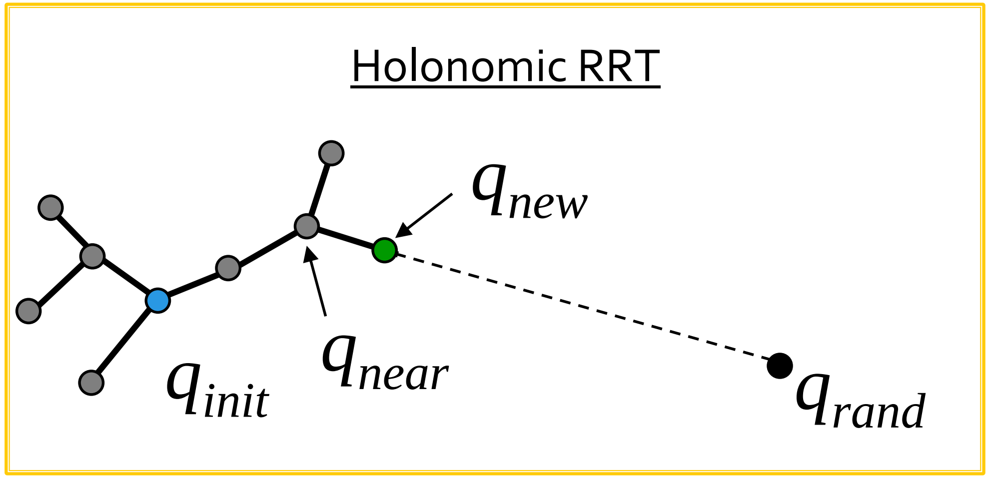
RRT-style Non-Holonomic Planning

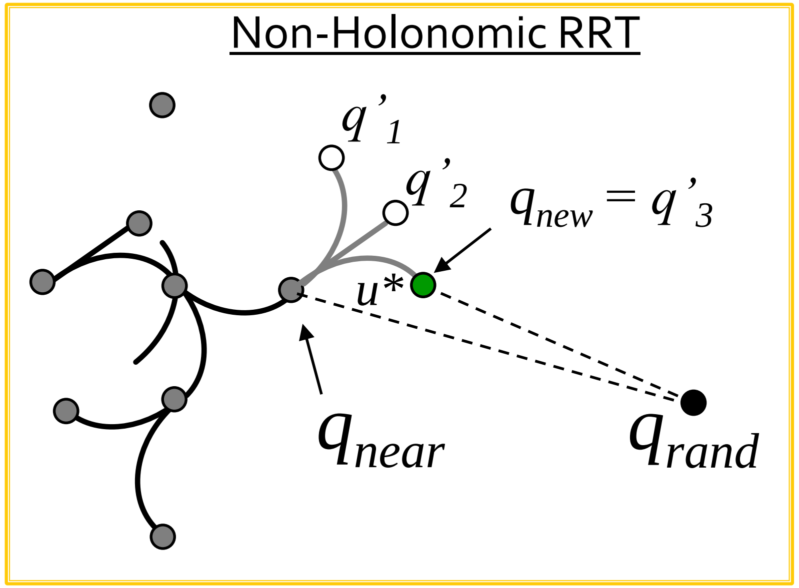
- RRT was originally proposed for non-holonomic planning
- Sampling and tree building is the same as regular RRT
- Additional concerns
  - Not all straight lines are valid, can't extend toward nodes
  - Use **motion primitives** to get as close to target node as possible

RRTs for Non-Holonomic Systems

- Apply motion primitives (i.e. simple actions) at q_{near}

$$q' = f(q,u) \text{ --- use action } u \text{ from } q \text{ to arrive at } q' \qquad \text{chose } u_* = \arg\min(d(q_{rand}, q'))$$



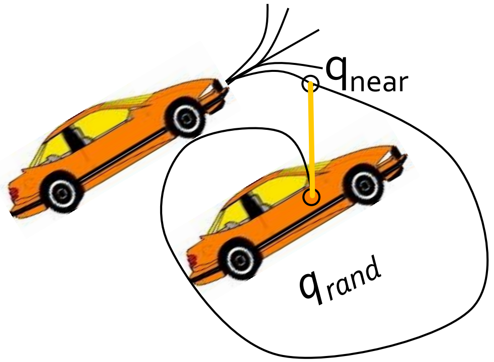


- You probably won't reach q_{rand} by doing this
  - Key point: No problem, you're still exploring!

RRTs and Distance Metrics

- Hard to define *d*, the distance metric
  - Mixing velocity, position, rotation ,etc.

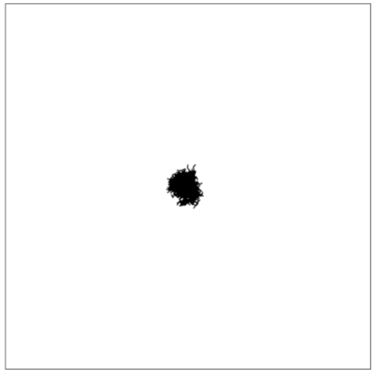
How do you pick a good q_{near}?



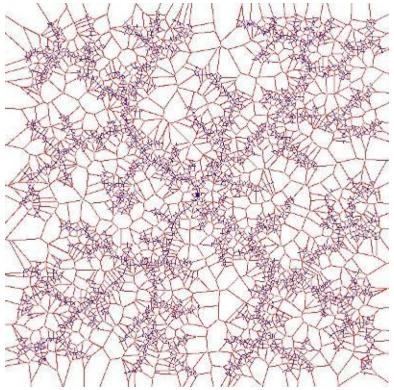
Configurations are close according to Euclidian metric, but actual distance is large

Introduce Voronoi bias to sampling

- At each iteration, the **probability that a node is selected is proportional to the volume of its Voronoi region**



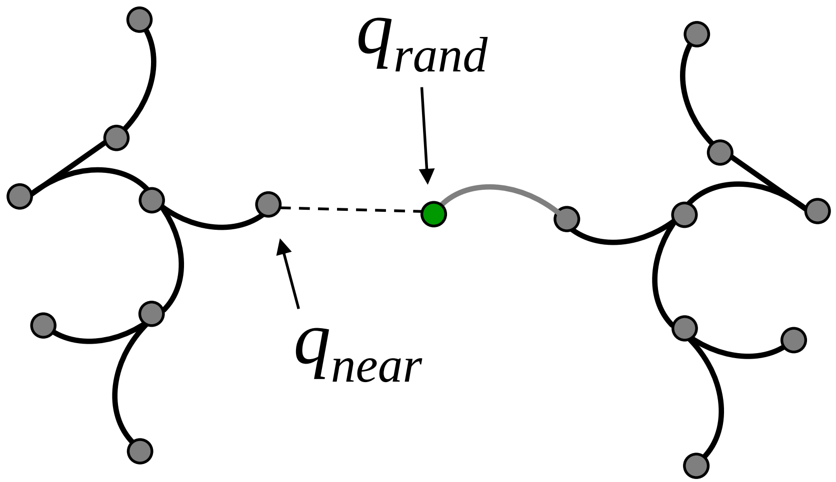
Random Node Choice
(bad distance metric)



Voronoi Bias
(good distance metric)

RRTs can rapidly expand toward region of large clearance!

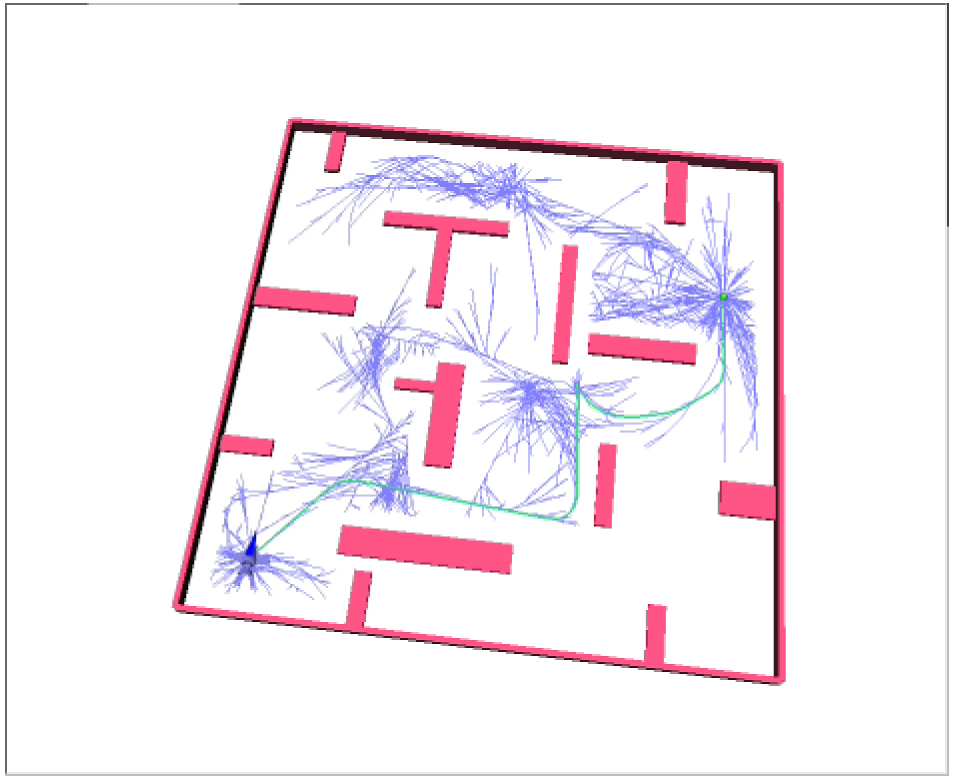
BiDirectional Non-Holonomic RRT



How to bridge between the two points?

Non-holonomic Smoothing

- Similar to holonomic case, paths produced can be highly suboptimal



Hovercraft with 2 Thrusters in 2D

Non-Holonomic Smoothing

- General trajectory optimization
- Convert path to cubic B-spline
  - Be careful about collisions

End