CS 4649/7649
Robot Intelligence: Planning

Motion Planning: Introduction & Reactive Strategies

Sungmoon Joo

School of Interactive Computing
College of Computing
Georgia Institute of Technology

*Slides based in part on Dr. Mike Stilman & Howie Choset's lecture slides

Course Info.

- TA - Saul
- HW#1 due next week
  - Trouble with group matching?
  - Need a repo.?
  - Trouble with install & running open-source planners?
  - Late policy
- Final Project Topic?

Motion Planning: Goal

- Motion planning is the ability for a robot to compute its own 'motions' in order to achieve certain goals.
  - (i) To compute 'motion strategies'
    - geometric path
    - time-parameterized trajectories
    - sequence of sensor-based motion commands
    - ...
  - (ii) To achieve high-level goals
    - go to A without colliding with obstacles
    - pick up the mug
    - ...
- All (autonomous) robots should eventually have this ability

Motion Planning: Sealing Process

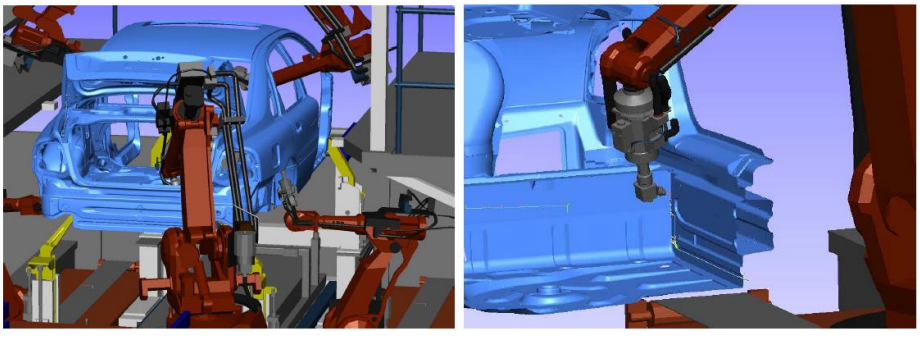


Figure from "Planning Algorithms" by Steven M. LaValle

Motion Planning: Piano Mover

Figure from "Planning Algorithms" by Steven M. LaValle

Motion Planning: Parking

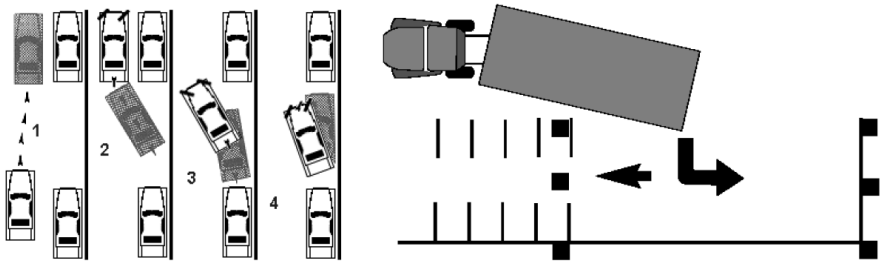
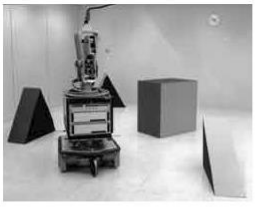


Figure from "Planning Algorithms" by Steven M. LaValle

Motion Planning: Goal

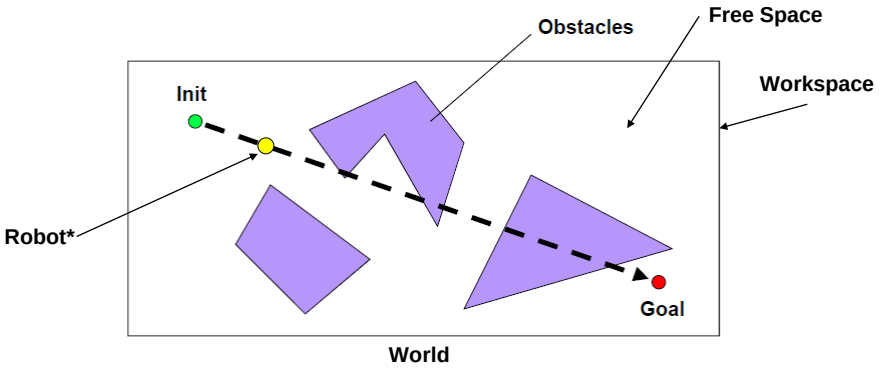


Make Decisions in Continuous State Spaces:

- Geometric
- Kinematic
- ...

Basic Path Planning Example

Problem: Find a continuous path (set of points) from Init to the Goal.



***Point robot**

Motion Planning Statement

"Compute a continuous sequence of collision-free robot configurations connecting the initial and goal configurations."

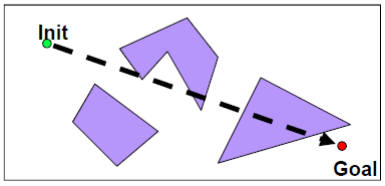
Notations/Definitions

- q: the robot's configuration, a complete specification of the position of every point in the system (a set of parameters that define the robot geometry in the world)
- W : the robot's workspace, the points the robot can possibly reach
- Wo_i : the i'th obstacle,
- W_{free} : the robot's freespace, $W_{free} = W - (\cup WO_i)$
- a path $c \in C^0$, $c : [0,1] \rightarrow W_{free}$ where c(0) is q_{init} and c(1) is q_{goal}

*Class C^0 : all continuous functions

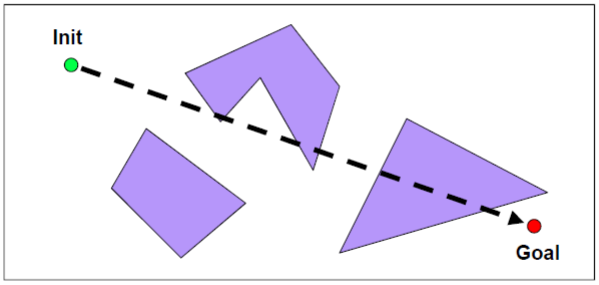
Completeness

- **Definition**
  - Finds a plan if one exists
  - Return failure in finite time otherwise
- **Importance**
  - **Gives guarantees for performance**
  - **Required in industry!**
  - **Suppose a task planner is using a motion planner to fulfill its actions...**



Optimality

Problem: Find the SHORTEST continuous path from Init to Goal.



Bug Algorithms (Lumelsky & Stepanov '87)

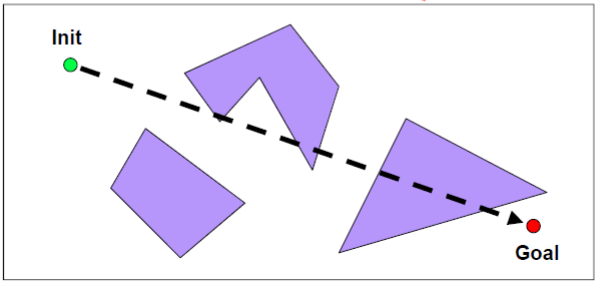
- Vast majority of planning algorithms assume global knowledge
- Bug algorithms: **local environment** knowledge & **global goal**
- **Robot can tell the distance (smell the goal)**
- Otherwise local sensing of walls
- Reasonable World
  - Finite obstacles in finite space
  - Workspace is bounded
- Bug behaviors (insect-inspired)
  - Follow a wall (right or left)
  - Move in a straight line toward goal

Bug Algorithms (Lumelsky & Stepanov '87)

~~• Full model?~~

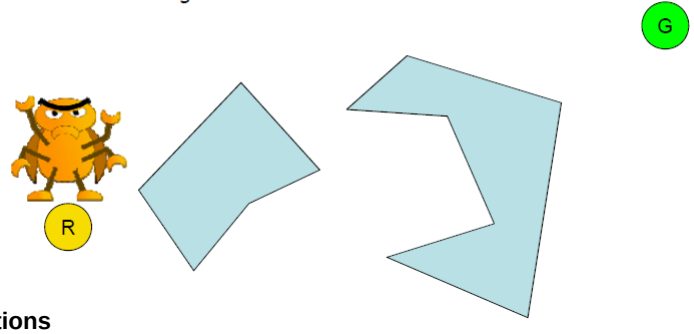
~~• Precise sensors?~~

~~• Planning algorithm?~~



Bug Algorithms

- Goal is known
- Obstacles are Unknown
- Limited Sensing

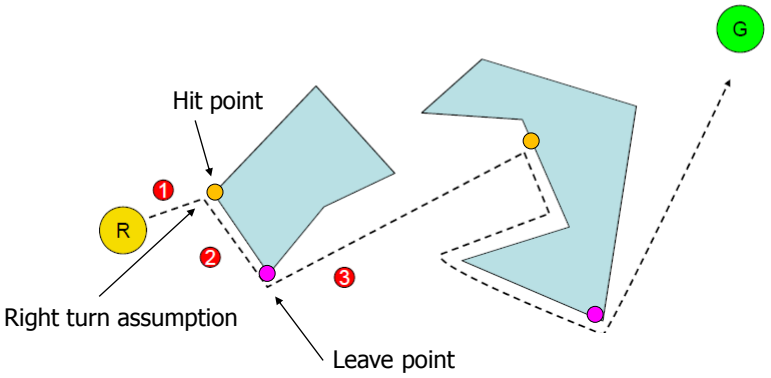


Assumptions

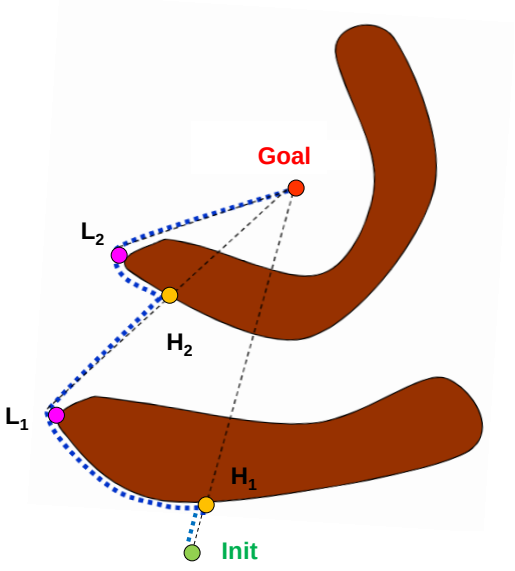
- **2D World. Point robot. Finite # of Obstacles. Obstacles have bounded perimeters**
- **Robot has not prior knowledge of locations, shapes of the obstacles**
- **Robot can senses its position with perfect accuracy. Goal is known.**
- **Robot can measure the distance btw two points. Robot can measure its traveled distance**
- **Robot can perfectly detect contact with an obstacle**

"Bug 0" Algorithm

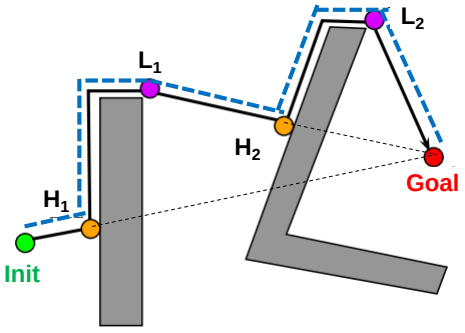
1. Move to Goal
2. Follow obstacle until you can go to goal again
3. Continue



"Bug 0" Algorithm



"Bug 0" Algorithm



"Bug 0" Algorithm

Pros:

- Solved the problem!
- No prior knowledge of environment
- Could use with prior knowledge if available

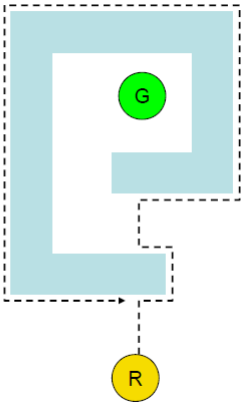
Cons

- Not Optimal
- Not Complete

"Bug 0" Algorithm

Pros:
- Solved the problem!
- No prior knowledge of environment
- Could use with prior knowledge if available

Cons
- Not Optimal
- Not Complete

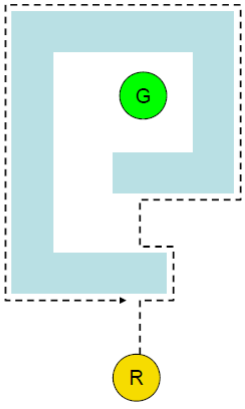


"Bug 0" Algorithm

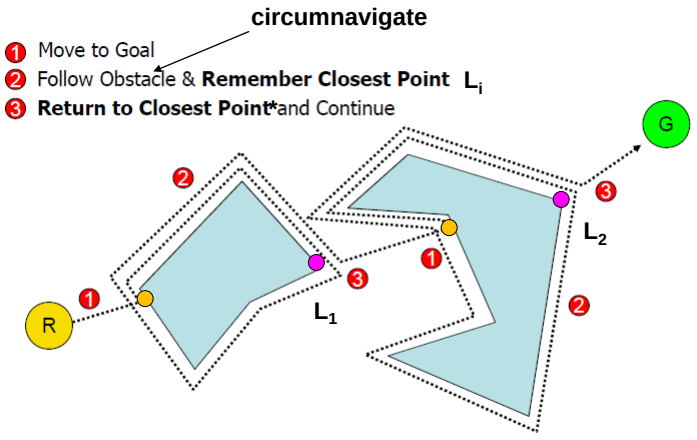
Pros:
- Solved the problem!
- No prior knowledge of environment
- Could use with prior knowledge if available

Cons
- Not Optimal
- Not Complete

Can We Make It Smarter?
What if it had MEMORY?



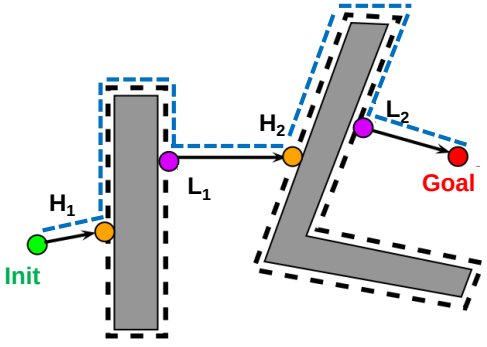
"Bug 1" Algorithm



Terminate when goal is found or no progress.

*By following the shortest path along the object boundary

"Bug 1" Algorithm



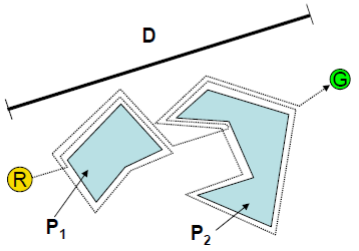
"Bug 1" Algorithm

D = Distance to Goal
P_i = Perimiter of Obstacle I

- Bounds on Path Length:
  - Minimum (Lower bound)

    D
  - Maximum (Upper bound)

    $D + 1.5 \sum_i P_i$



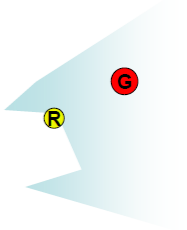
"Bug 1" Algorithm

Complete?
- **YES!**

Algorithm:
1. Move to Goal
2. Follow Obstacle & Remember Closest Point
3. Return to Closest Point and Continue

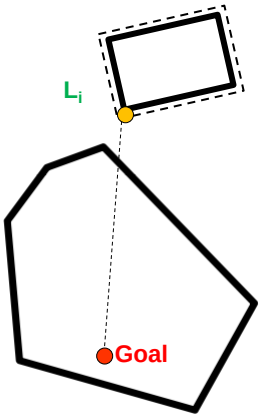
- Proof (Sketch):
  - **Leave point** = closest point on perimeter to Goal (1 point per object)
  - **Hit point** = point achieved after leaving a Leave point
  - The Algorithm Terminates
    - Each Leave **gets closer** than Hit
    - Each Hit **gets closer** than Leave
    - Finite number of (Hit, Leave) pairs

    Why?
  - Terminates without finding solution?
    - **LEAVE = Hit**
    - Goal is inside Obstacle



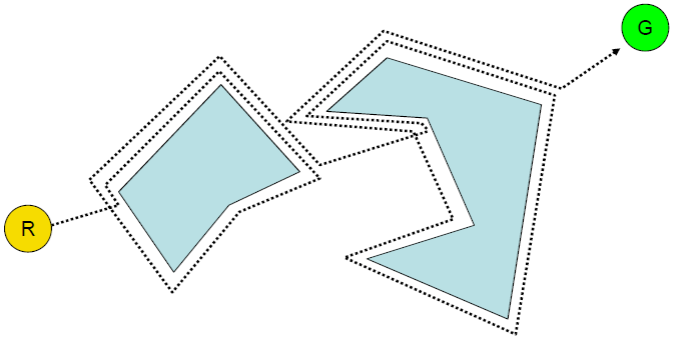
"Bug 1" Algorithm

How Can "Bug 1" Recognize that the goal is not reachable?



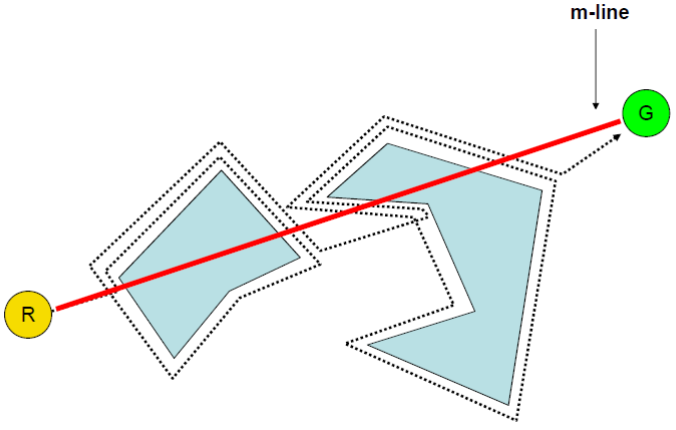
If the direction from L_i toward the goal points into the obstacle, then the goal can't be reached.

Improvements?



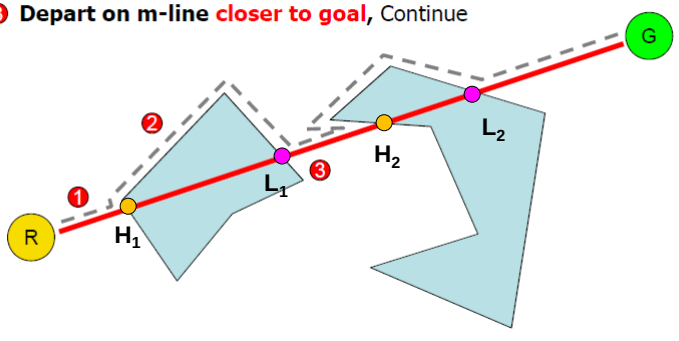
Improvements?

m-line: The straight line from the initial configuration to the goal configuration



"Bug 2" Algorithm

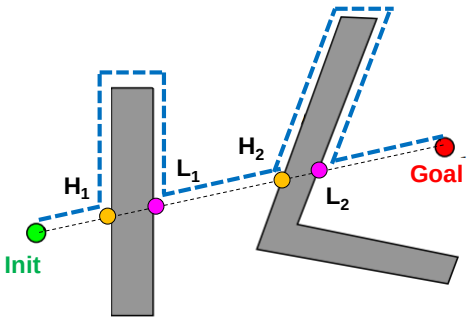
1. Move to Goal (following m-line)
2. Follow Obstacle* to m-line (closer to goal)
3. **Depart on m-line closer to goal,** Continue



Terminate when goal is found or no progress.

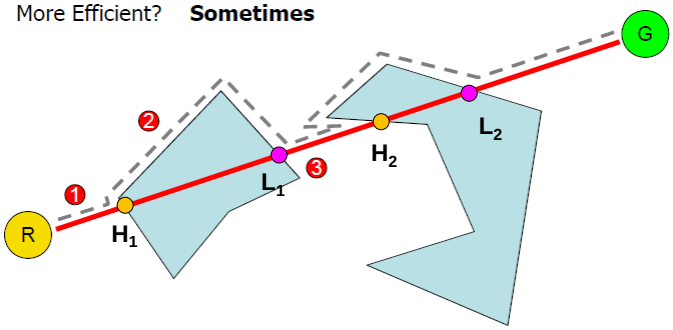
*Usually toward the left

"Bug 2" Algorithm



"Bug 2" Algorithm

- Complete? YES
- More Efficient? **Sometimes**



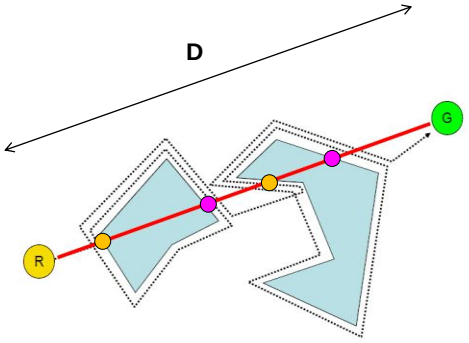
"Bug 2" Algorithm

D = Distance to Goal
P_i = Perimiter of Obstacle I

- Bounds on Path Length:
  - Minimum

    D
  - Maximum

    $D + \sum_i \frac{n_i}{2} P_i$

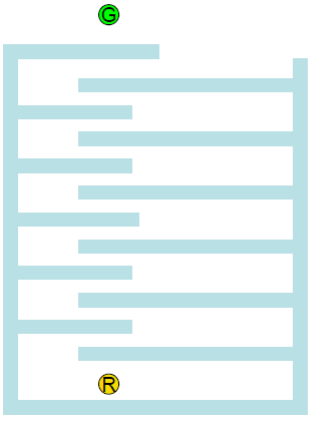


n_i = # of intersections of the i^{th} obstacle with m-line

Comparison of Bug 1 & Bug 2

Bug 2 Wins!

Bug 1 Wins!



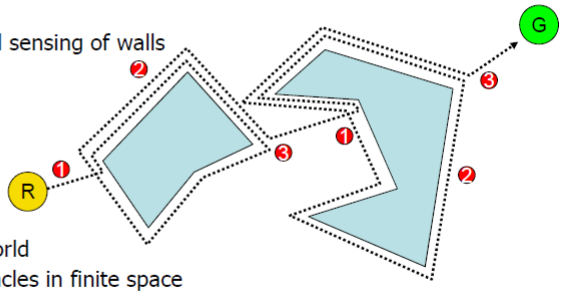
Comparison of Bug 1 & Bug 2

- Bug 1
  - Exhaustive Search
  - Evaluates Choices before choosing
- Bug 2
  - Greedy, Heuristic Algorithm
- Mostly, Bug 2 performs better and Bug 1 is more predictable
- Both are complete, neither is optimal!
- There exist variants for more complex sensors (see Tangent Bug)

Bug Algorithms Summary

local environment knowledge & **global goal**
Robot can tell the distance (smell the goal)

- Otherwise local sensing of walls



- Reasonable World
  - Finite obstacles in finite space
  - Workspace is bounded

Bug Algorithms Summary

Completeness is Desirable

- Completeness does not always require complexity
- Bug Algorithms achieve global completeness with local planning

Optimality is Desirable

- Bug Algorithms are not Optimal

Bug Algorithms Summary

Algorithm	Bug 0	Bug 1	Bug 2
Completeness	X	0, Exhaustive	0, Greedy
Characteristic	-	Safe, Reliable	Better in some cases. But worse in other cases

***None of them is optimal**

Planning Requires Models

• Bug algorithms don't plan ahead.

• They are not really motion planners, but "reactive motion strategies"

• To plan its actions, a robot needs a (possibly imperfect) predictive model of the effects of its actions, so that it can choose among several possible combinations of actions

A Useful Notion: Competitive Ratio (CR)

• Bug algorithms are examples of online algorithms where a robot discovers its environment while moving

• Competitive Ratio: To evaluate algorithms that utilize different information
- An *on-line algorithm vs* an algorithm that receives more information
- The competitive ratio of an online algorithm A is the maximum over all possible environments of the ratio of the length of the path computed by A by the length of the path computed by an optimal offline algorithm B that is given a model of the environment

$$\max_{e \in E} \frac{\text{Cost of executing the plan that does not know } e \text{ in advance.}}{\text{Cost of executing the plan that knows } e \text{ in advance}}.$$

A Useful Notion: Competitive Ratio (CR)

Lost Cow Problem

a short-sighted cow is following along an infinite fence and wants to find the gate

(a)If the cow is told that the gate is exactly distance 1unit away
CR?
(b)If the cow is told only that the gate is at least distance 1unit away
CR?

Figure from "Planning Algorithms" by Steven M. LaValle