15-494/694: Cognitive Robotics

Dave Touretzky

Lecture 9:

Path Planning with Rapidly-exploring Random Trees

Navigating with the Pilot

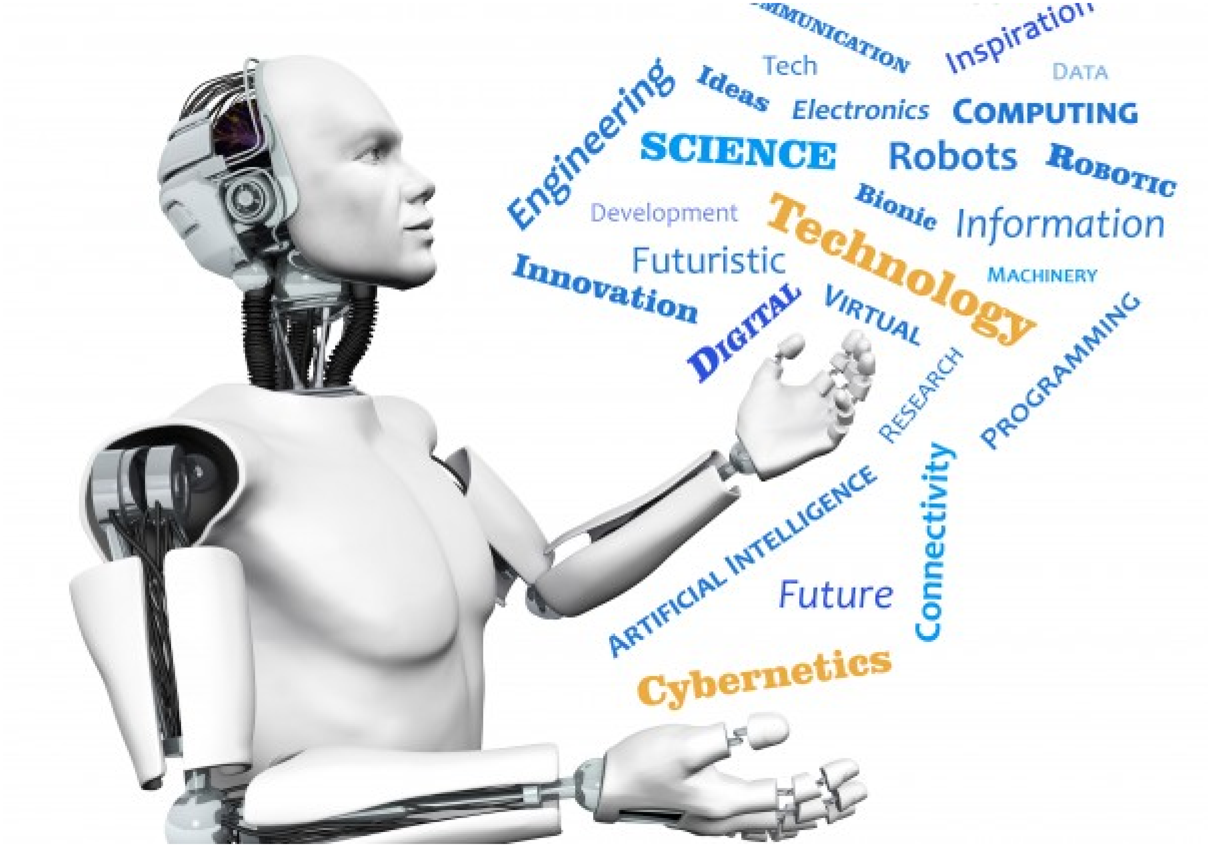


Image from http://www.futuristgerd.com/2015/09/10

Outline

- How is path planning used in robotics?
- Path planning as state space search
- RRTs: Rapidly-exploring Random Trees
- The RRT-Connect algorithm
- Collision detection
- Smoothing
- Path planning with constraints
- Navigating with the Pilot

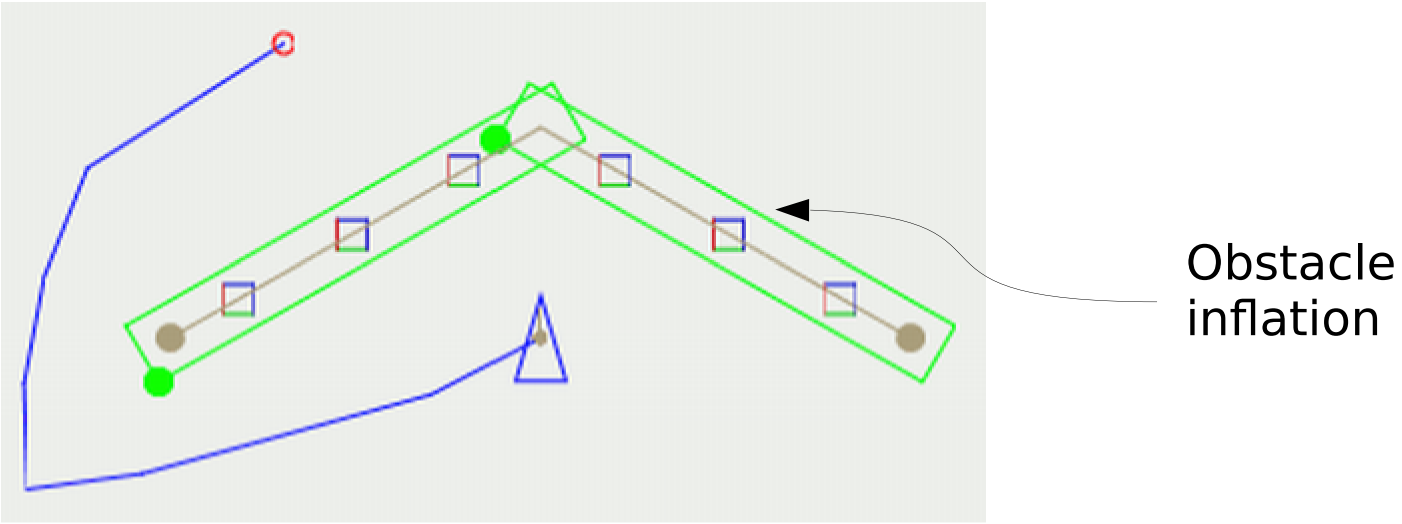
Path Planning in Robotics

1. Navigation path planning
   - How to get from the robot's current location to a goal.
   - Avoid obstacles.
   - Provide for localization.

2. Manipulation path planning
   - Move an arm to grasp and manipulate an object.
   - Avoid obstacles.
   - Obey constraints (e.g., don't spill the coffee).

Navigation Planning

- 2D state space: (x,y) coordinates of the robot
  - Treat the robot as a point or a circle.



- 3D state space: (x,y,θ) pose of the robot
  - Heading matters when the robot is asymmetric
  - Heading matters when the robot's motion is constrained

Grid-Based Path Planning

- Discretizes the environment into a 2D grid.
- Can use best-first or A* search.
- Works okay in small spaces.

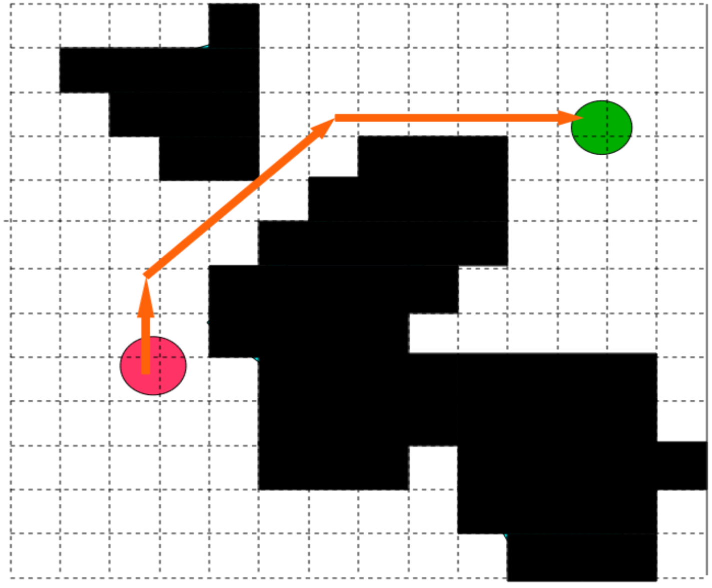
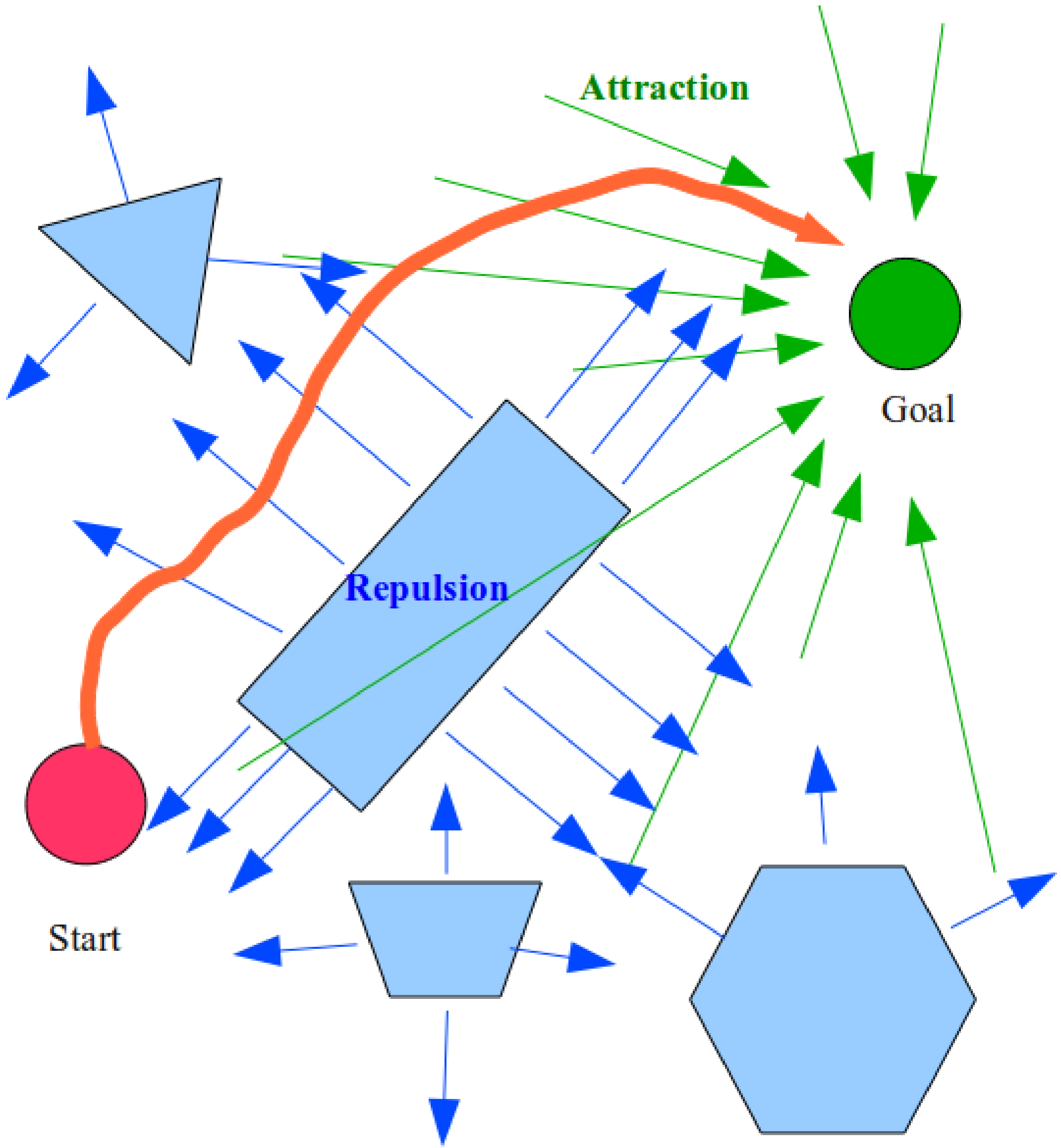


Figure from http://www.gamasutra.com/blogs/MattKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

But it has its drawbacks:

- Treats the robot as a point. Unrealistic!
- Not efficient in higher dimensional state spaces.

Potential Field Path Planning



- Can fail due to local minima in the potential function.
- Consider a U-shaped obstacle.
- Requires careful tuning.

Figure from http://www.gamasutra.com/blogs/MattKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Cspace Transform

- The area around an obstacle that would cause a collision with the robot.

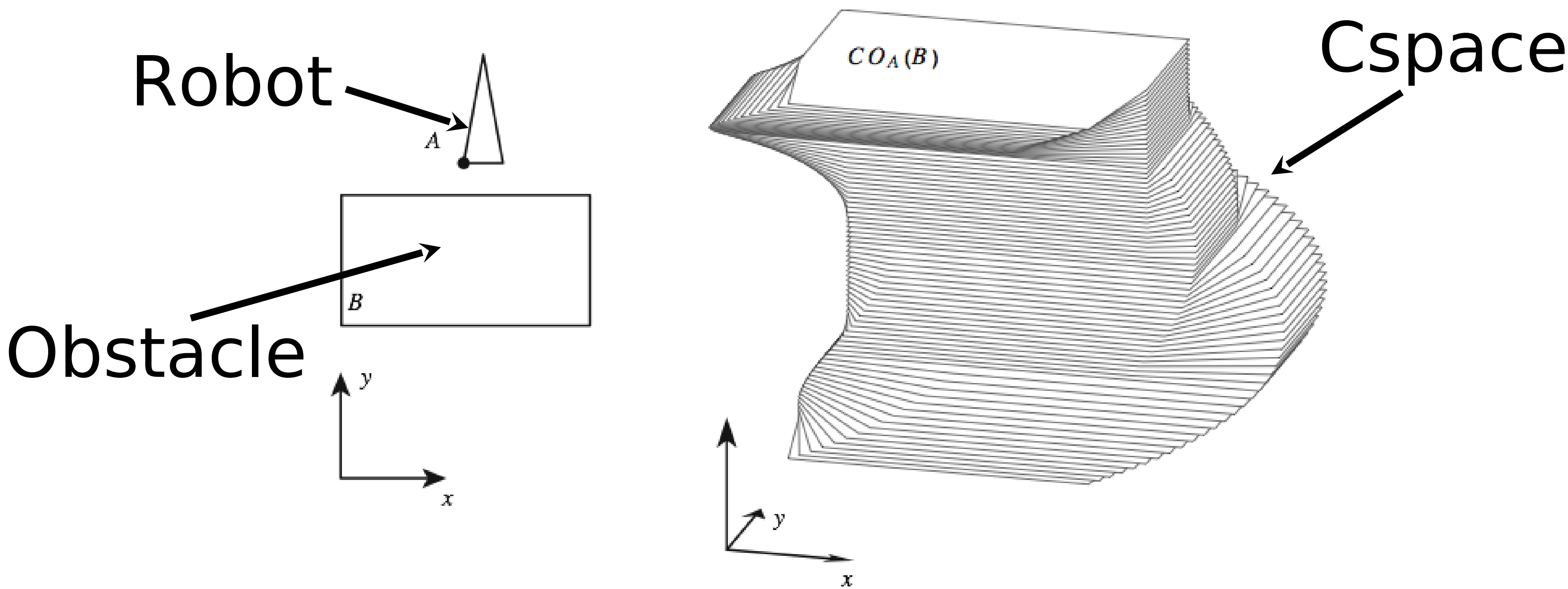


Figure 4.4 - Mason, Mechanics Of Robotic Manipulation

Arm Path Planning

- Cspace transform blocks out regions of joint space

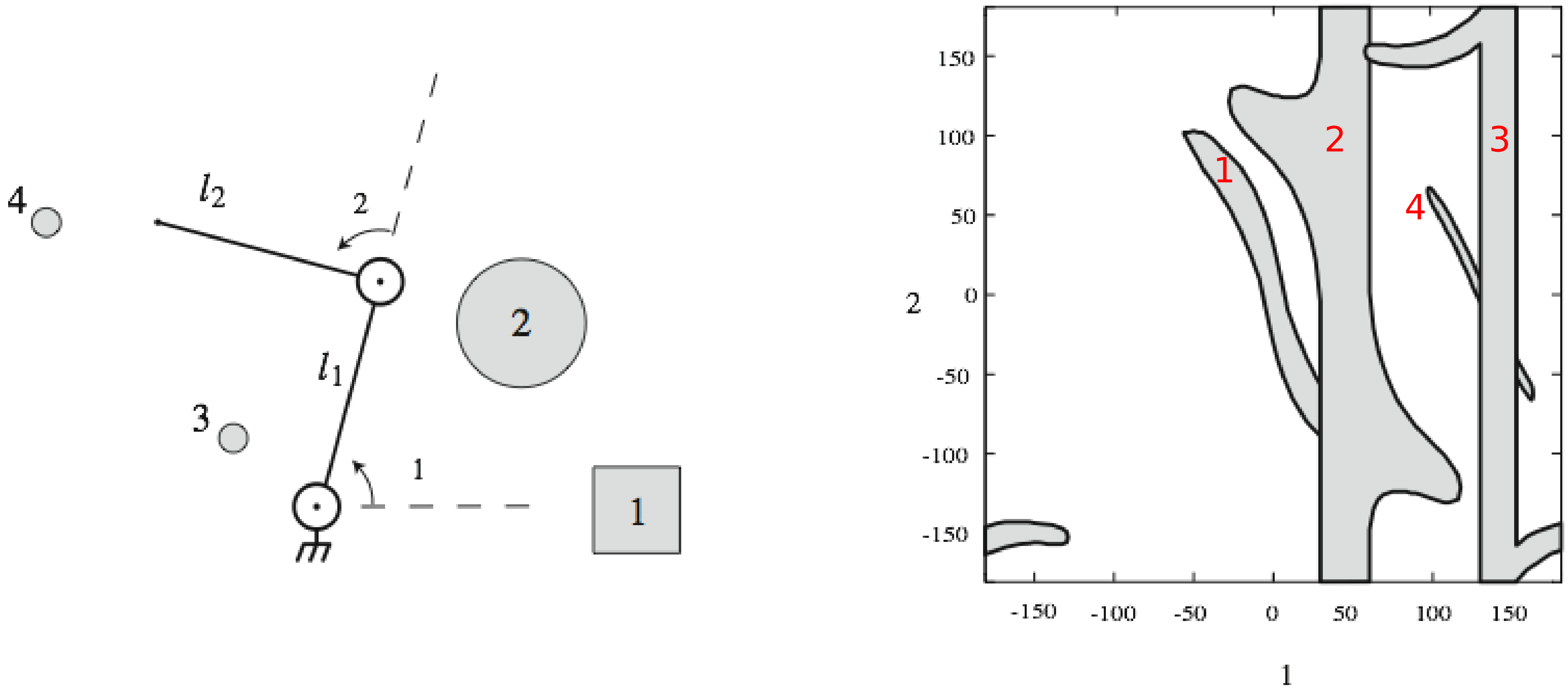


Figure 4.5 - Mason, Mechanics Of Robotic Manipulation

State Space Search

The path planning problem:

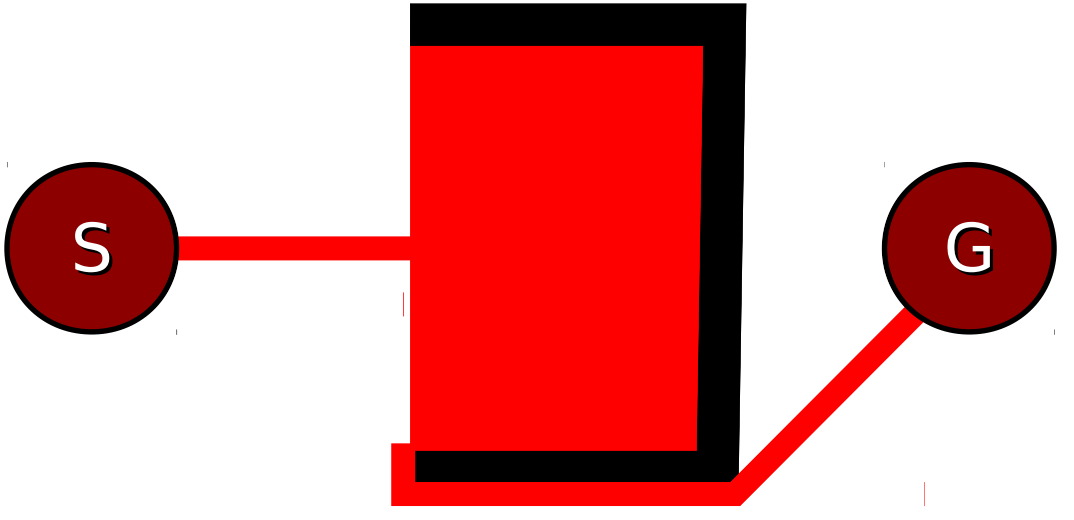
Given an n-dimensional state space and

- a start state $S=[s_1,s_2,\ldots,s_n]$
- a goal state $G=[g_1,g_2,\ldots,g_n]$
- an admissibility predicate P (collision test + constraints)

find a path from S to G such that every state on the path satisfies P.

Best First or A* Search Can Be Slow

- Can get trapped in a cul de sac for a long time.



- See search animation videos on YouTube.
- Random search might be faster.

Rapidly-exploring Random Trees

- Described in LaValle (1998), Kuffner & LaValle (2000)
- Create a tree with start state S as the root.
- Repeat up to K times:

  Pick a point $\mathbf{q}_{rand}$ in configuration space:

  - Sometimes $\mathbf{q}_{rand}$ is really random
  - Sometimes $\mathbf{q}_{rand}$ is the goal G

  – Find $\mathbf{q}_{nearest}$, the closest node to $\mathbf{q}_{rand}$

  - Add a new node $\mathbf{q}_{new}$ by extending $\mathbf{q}_{nearest}$ some distance Δ toward $\mathbf{q}_{rand}$.

  - If $\mathbf{q}_{new}$ is close enough to the goal G, return.

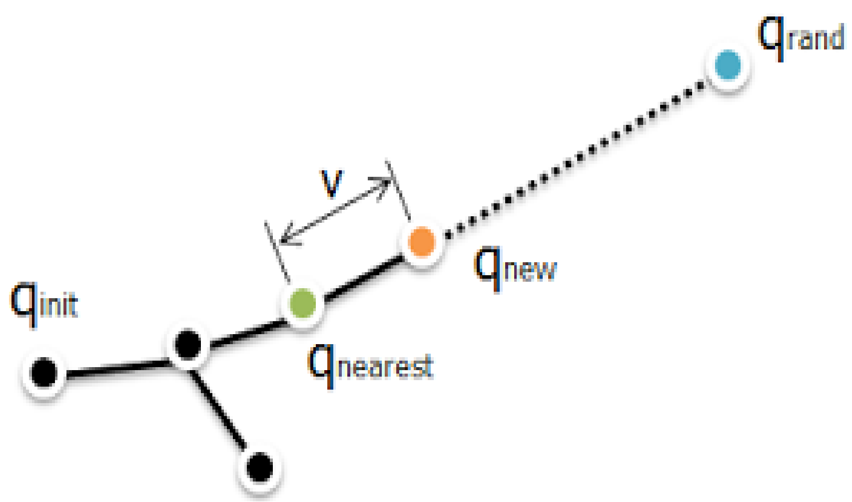
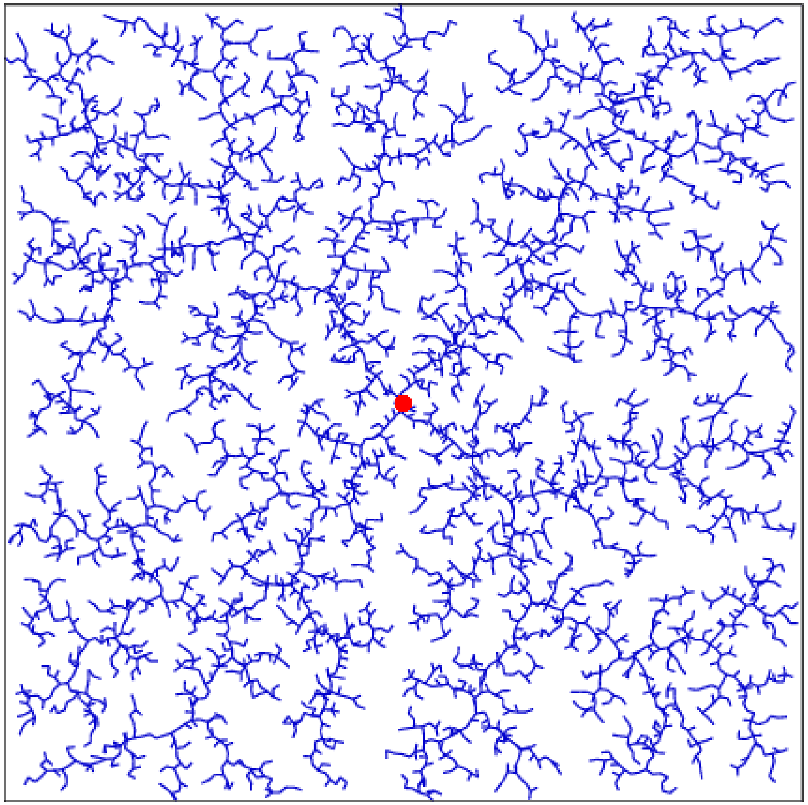


Image from http://joonlecture.blogspot.com/2011/02/improving-optimality-of-rrt-rrt.html

RRT Algorithm

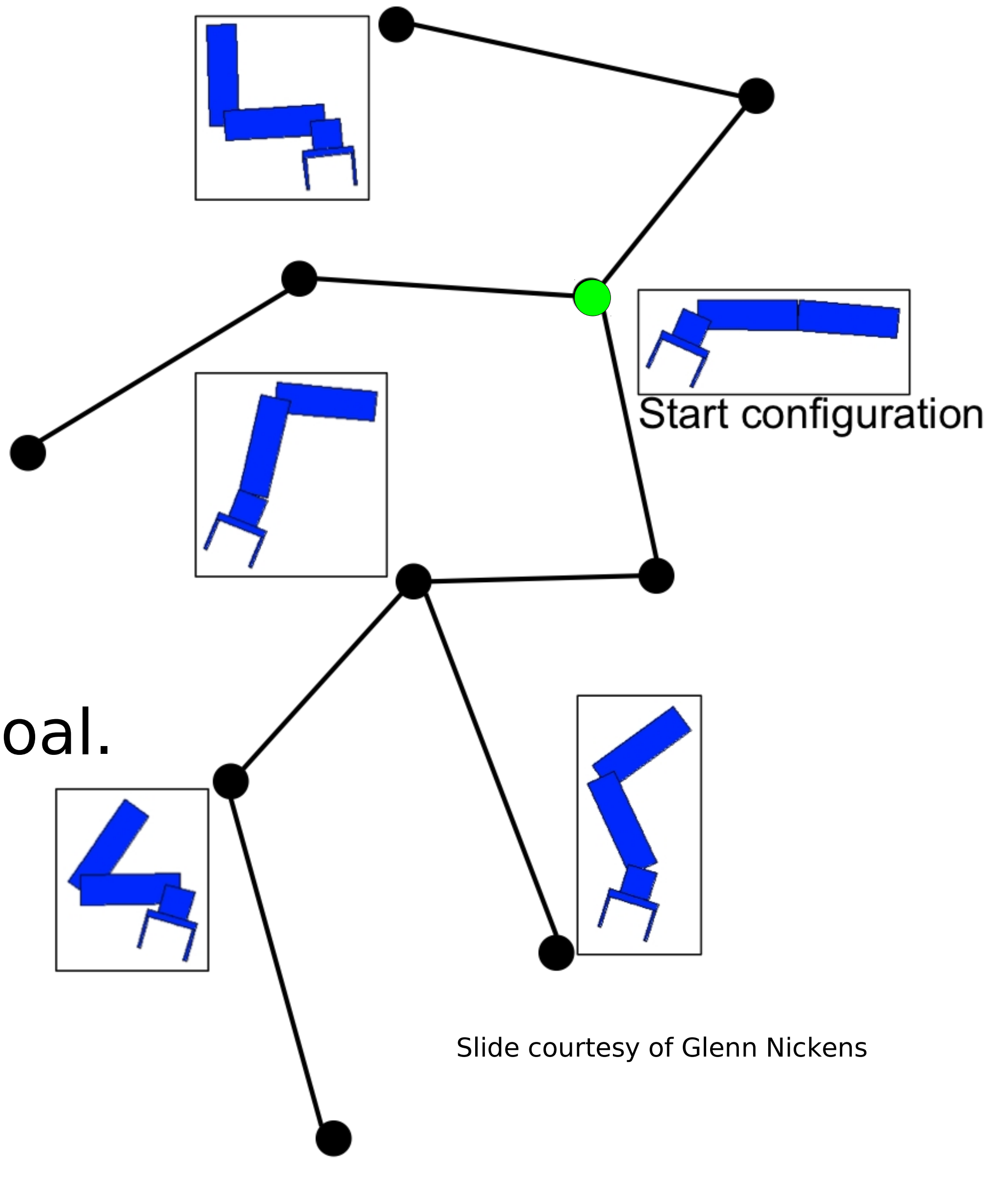
- Rapidly samples the state space.
- Cannot get trapped in local minima.
- Works well in high-dimensional spaces.
- Does not generate smooth paths.
- Can't tell when no solution exists; only quits when it exceeds the iteration limit K.



http://msl.cs.uiuc.edu/rrt/treemovie.gif

RRTs for Arm Path Planning

- Each node encodes an arm configuration in joint space.
- Only add nodes that don't cause collisions (with self or obstacles).
- Alternately (i) extend the tree in random directions and (ii) move toward the goal.



Slide courtesy of Glenn Nickens

Implementation Notes

- Finding $\mathbf{q}_{\text{nearest}}$, the nearest node in the tree to $\mathbf{q}_{\text{rand}}$, is the most expensive part of the algorithm.
  - Use K-D trees to efficiently find $\mathbf{q}_{\text{nearest}}$?
  - In practice, K-D trees are slower unless you have a huge number of nodes (several thousand).

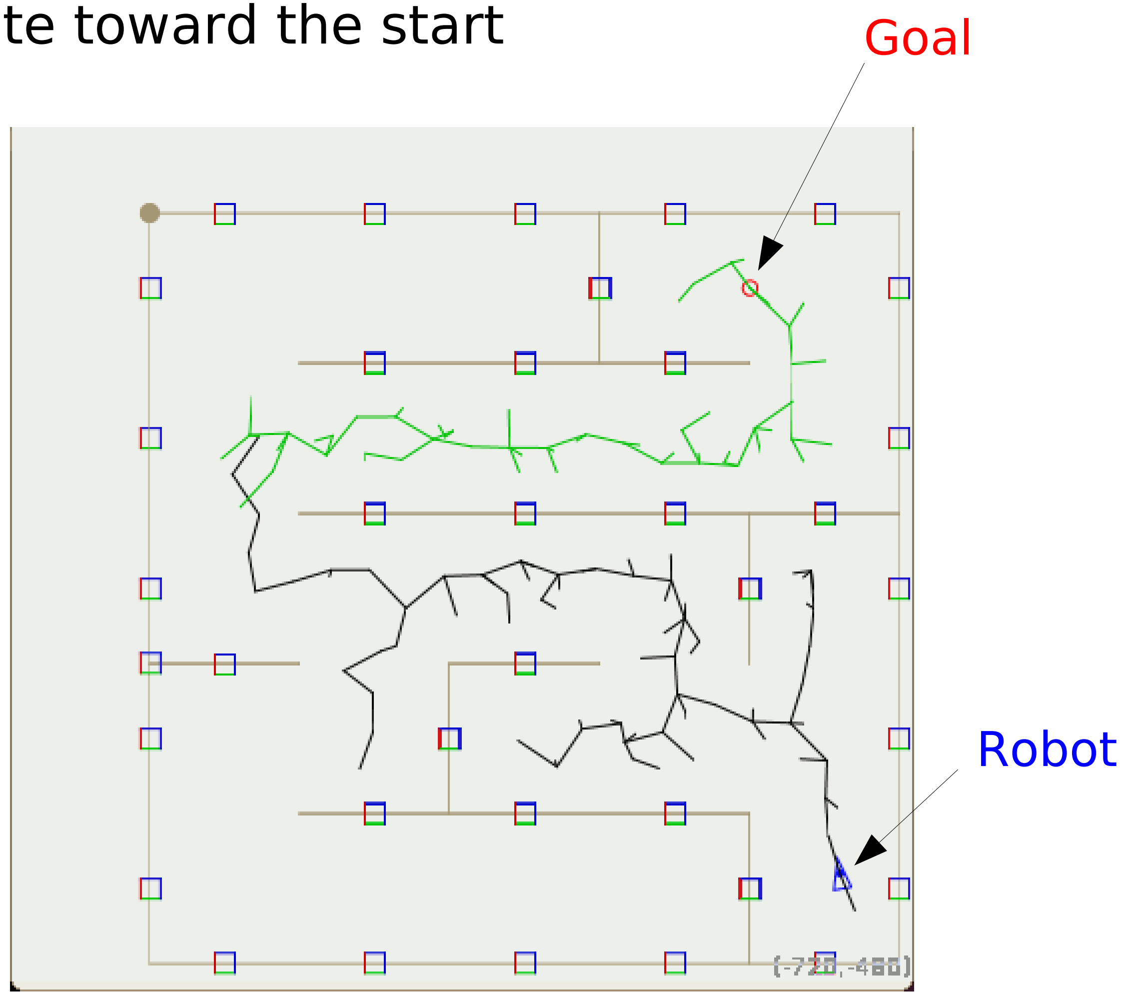
- Why only go a distance Δ toward the goal state G? Why not go as far as we can, in steps of Δ?
  - With no obstacles, this reaches the goal very quickly, but random search will get there nearly as quickly as we keep extending the nearest node to the goal.
  - But when obstacles are present, this can waste time filling out branches that will ultimately fail.
  - Generating lots of extra nodes bloats the tree, which slows down the algorithm.

RRT-Connect Algorithm

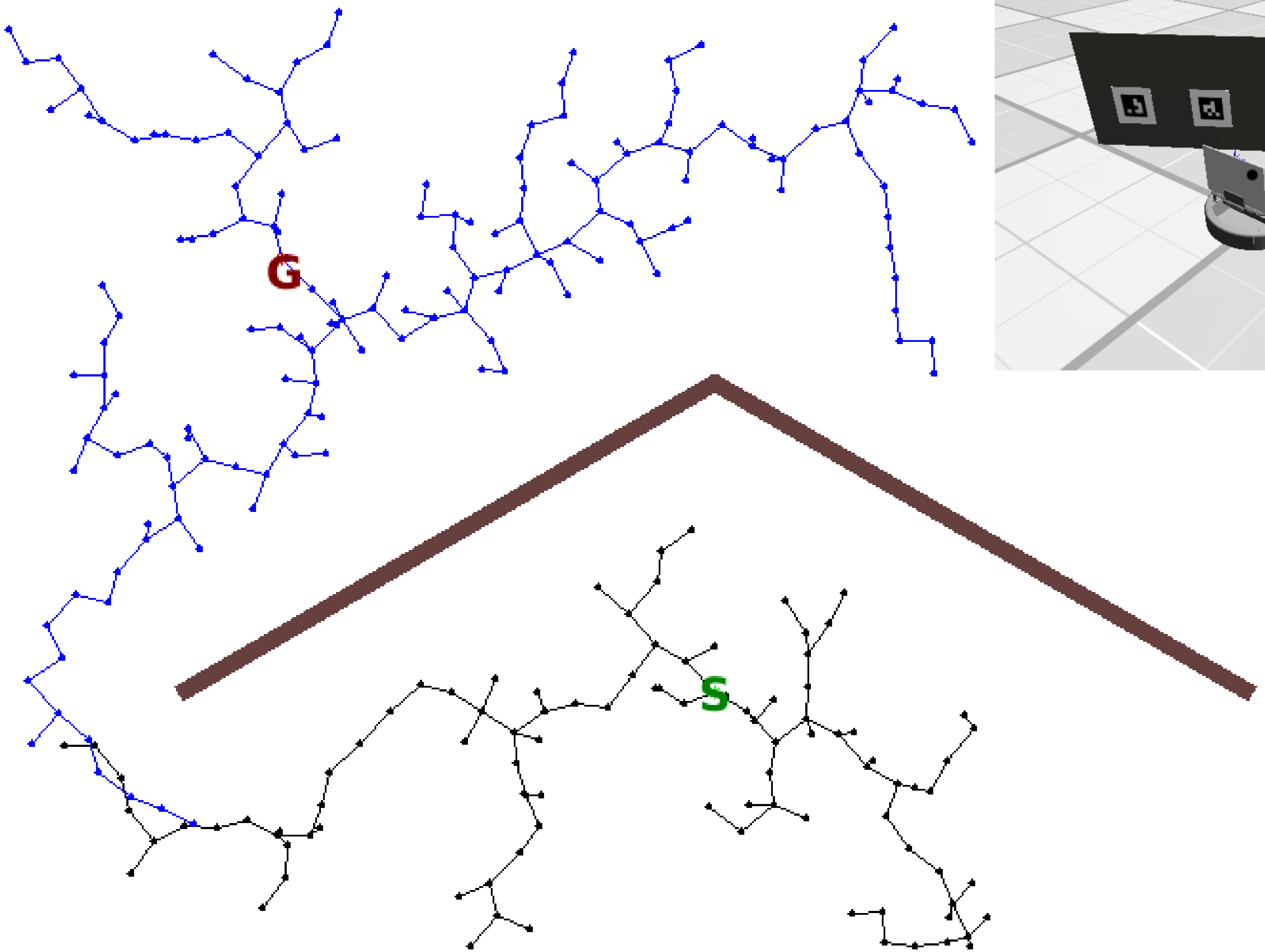
- Variant of RRT that grows two trees:
  - one from the start state toward the goal
  - one from the goal state toward the start

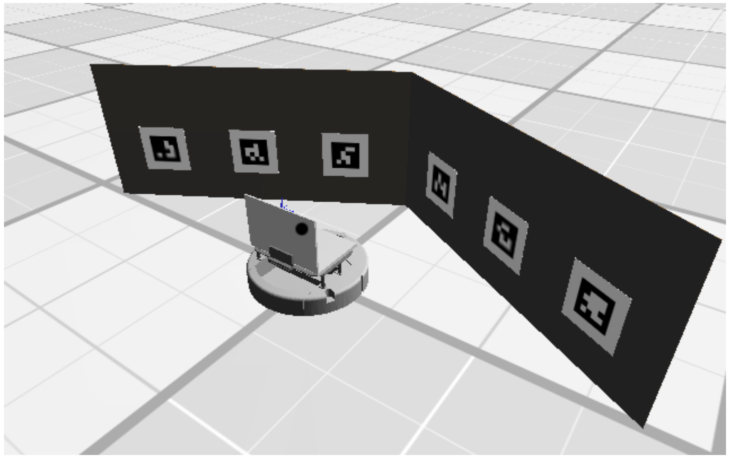
- When the two trees connect, a solution has been found.

- Not guaranteed to be better than RRT, but often helps.



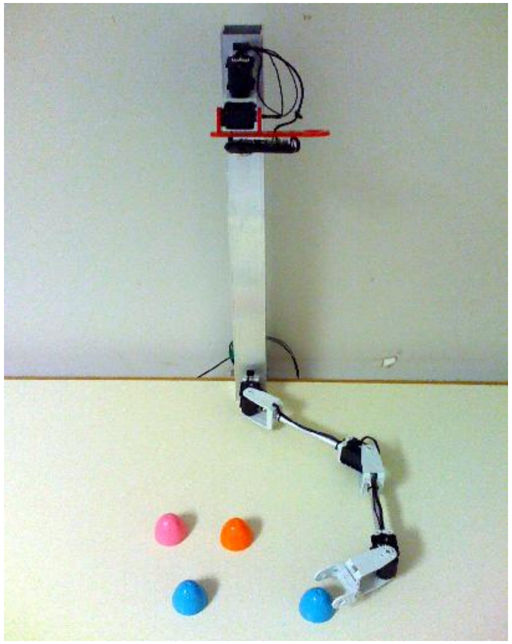
RRTs in An Open Field

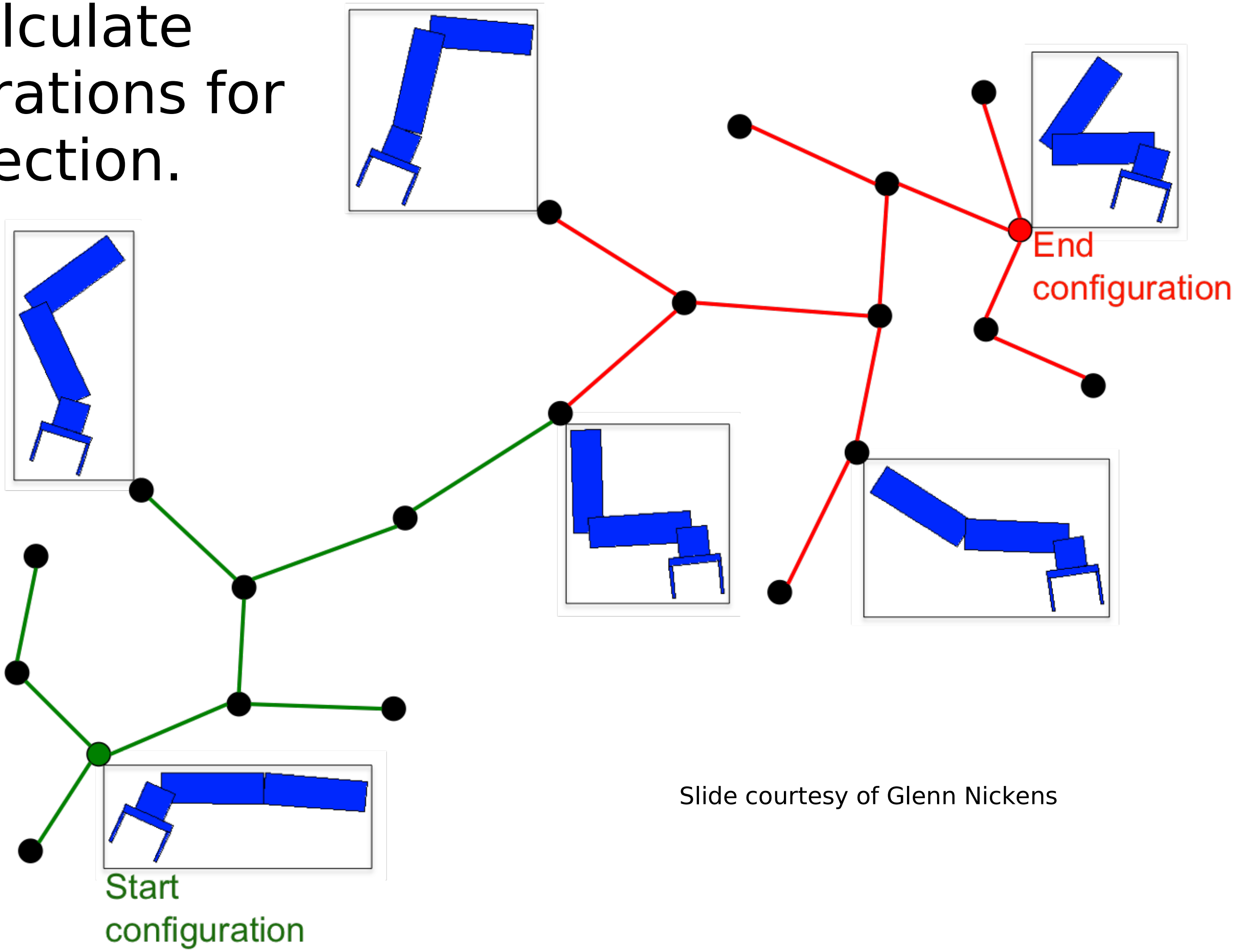




RRT-Connect For Arms

- Use IK to calculate the goal configuration.
- Use FK to calculate arm configurations for collision detection.





Slide courtesy of Glenn Nickens

Collision Detection

- Represent the robot and the obstacles as **convex polygons**.
- In 2D, use the Separating Axis Theorem to check for collisions.
    - Easy to code
    - Fast to compute
- In 3D, things get more complex.
    - Tekkotsu uses the GJK (Gilbert-Johnson-Keerthi) algorithm, used in many physics engines for video games.

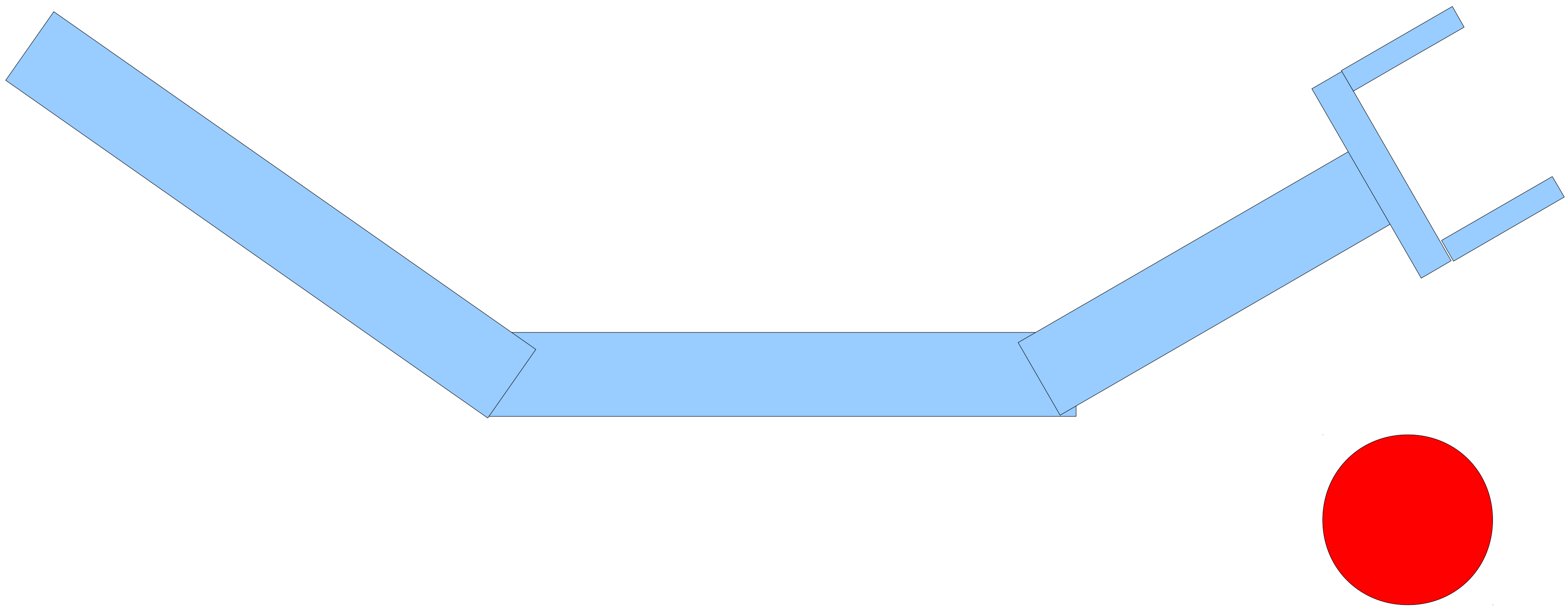
Algorithm to Apply the SAT

- For every edge of polygon A and of polygon B:
  - Project all the vertices onto the line normal to that edge.
  - Calculate the min and max coordinates for each polygon
  - If minA $<$ minB and maxA $>$ minB OR
    if minB $<$ minA and maxB $>$ minA
    then the polygons collide.

- If you find any edge projection in which the ranges don't overlap, the polygons do not collide.

Arm Collision Detection

- Represent each link as a separate polygon.
- Check for:
  - Self-collisions other than link *n* with link *n+1*
  - Collisions of a link with an obstacle

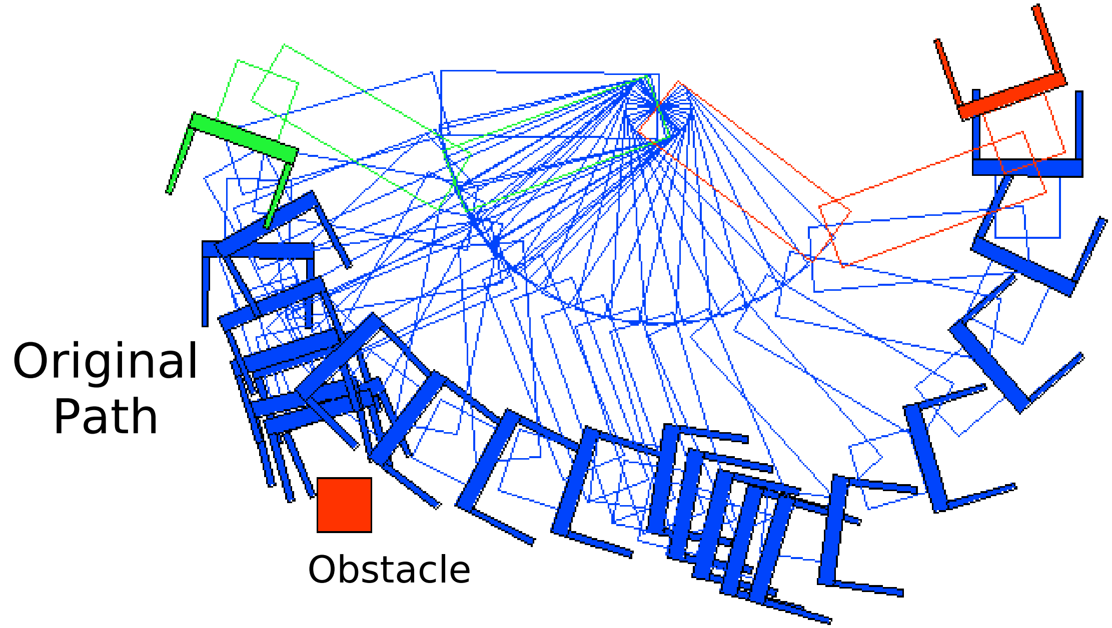


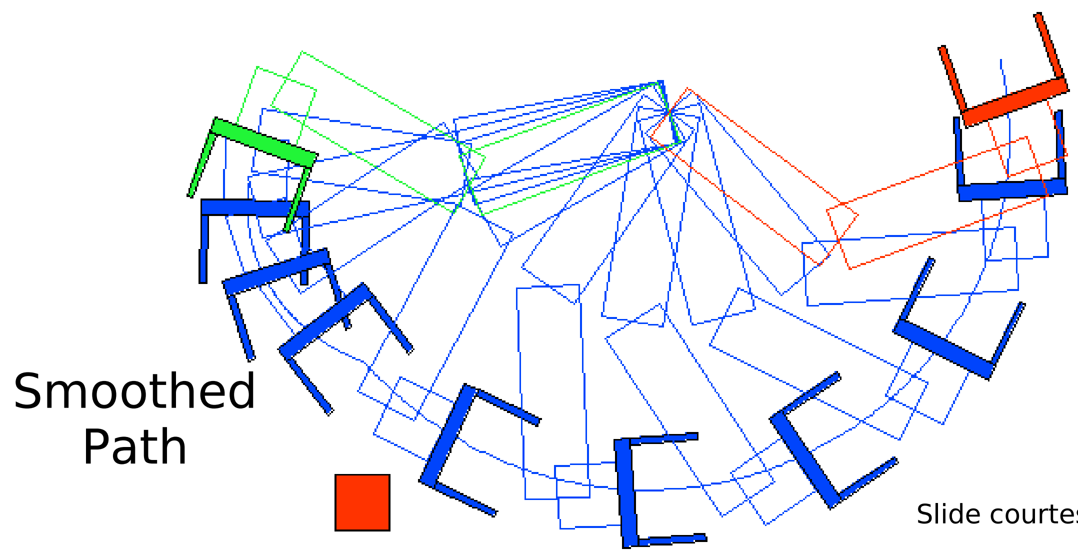
Path Smoothing

- The random component of RRT-Connect search often results in a jerky and meandering solution.
- Solution: apply a path smoothing algorithm.
- Repeat N times:
  - Pick two points on the path at random
  - See if we can linearly interpolate between those points without collisions.
  - If so, then snip out that segment of the path.

Smoothing An Arm Trajectory

- Start state
- Intermed. states
- End state

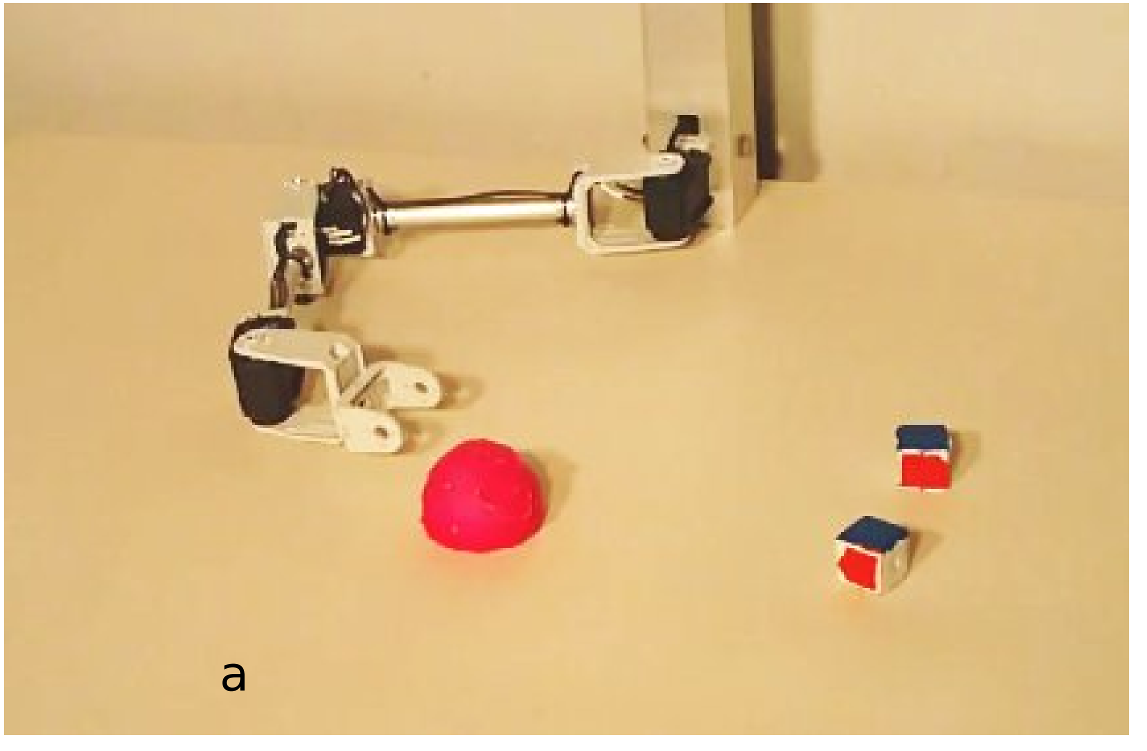




Slide courtesy of Glenn Nickens

Path Planning With Constraints

- With no closeable fingers, arm motion is constrained to be within about 60° of finger direction or we'll lose the object.



(video)

http://www.youtube.com/watch?v=9oDQ754YVoc

Implementing Constraints

- Each time we generate a new state $\mathbf{q}_{new}$:
  - Check to see if $\mathbf{q}_{new}$ obeys the constraint.
  - For finger motion constraint, check if the direction of motion from parent state $\mathbf{q}_{nearest}$ to new state $\mathbf{q}_{new}$ is within 60° of the finger direction.
- What if $\mathbf{q}_{new}$ doesn't obey the constraint?
  - Reject it and pick a new $\mathbf{q}_{rand}$ from which we'll generate a new $\mathbf{q}_{new}$.
  - Or try to “fix” $\mathbf{q}_{new}$ by perturbing its value slightly so as to satisfy the constraint.

Path Planning Failure

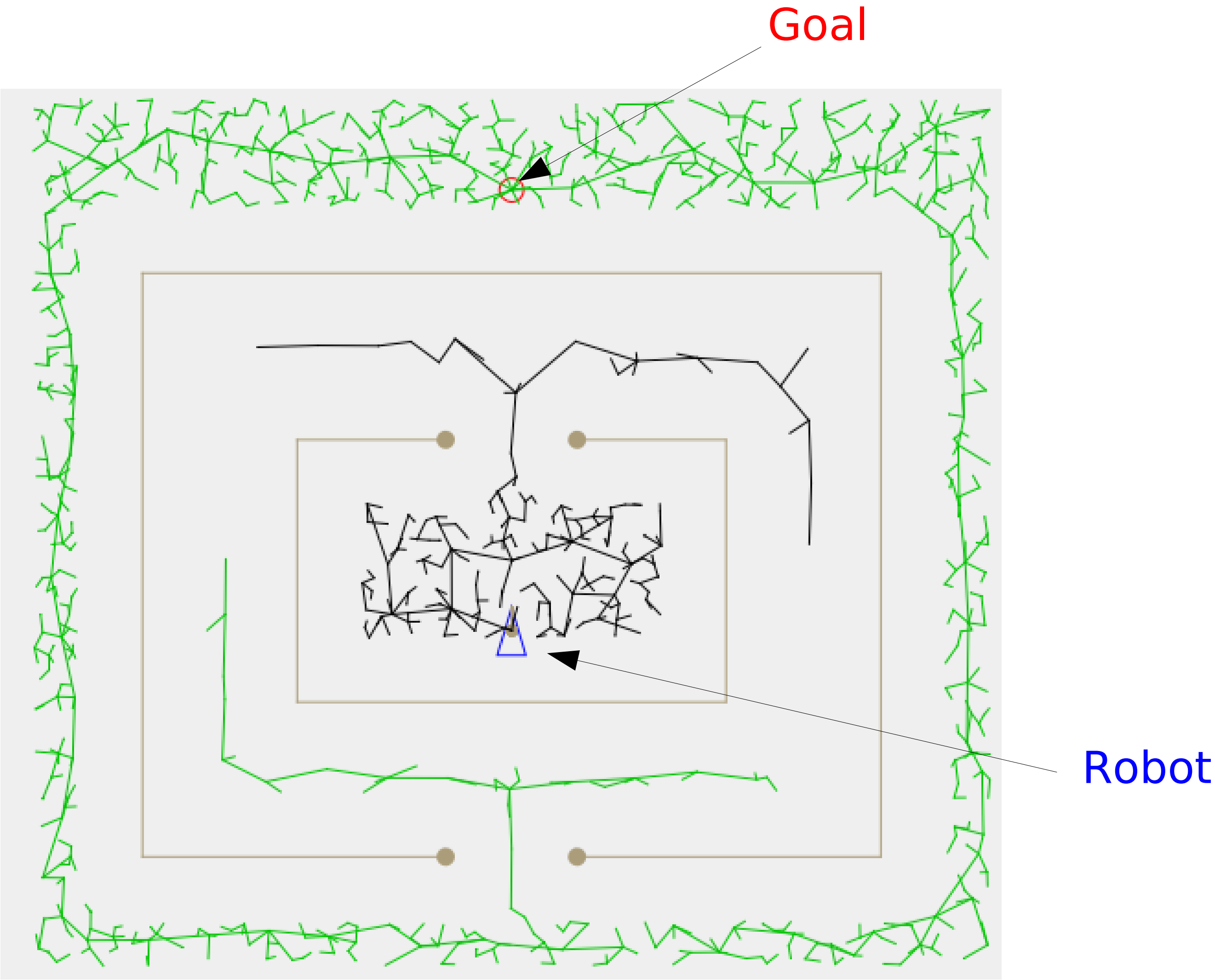
RRT path planning can legitimately fail if:

- There is no route to the goal due to obstacles blocking every path from start to goal.
- The paths to the goal don't lie entirely within the allowed world bounds (world map too small).

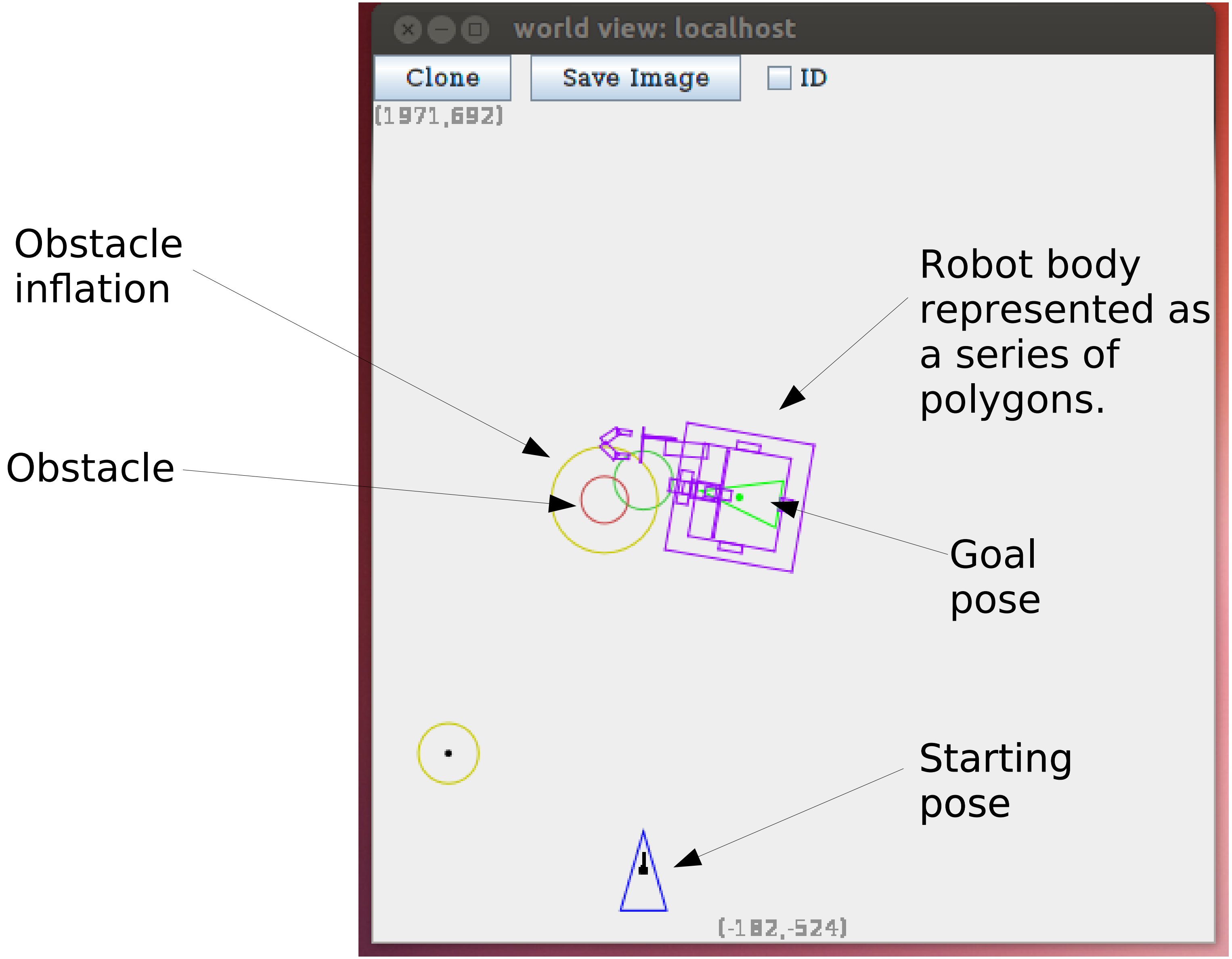
But it can also fail if:

- The iteration limit was set too low.
- The start state is already in collision with something.
- The goal state is in collision with something.

Running Out of Iterations



Path Planning Failure: Goal State Is In Collision



Full 3D Path Planning:
The Piano Movers Problem

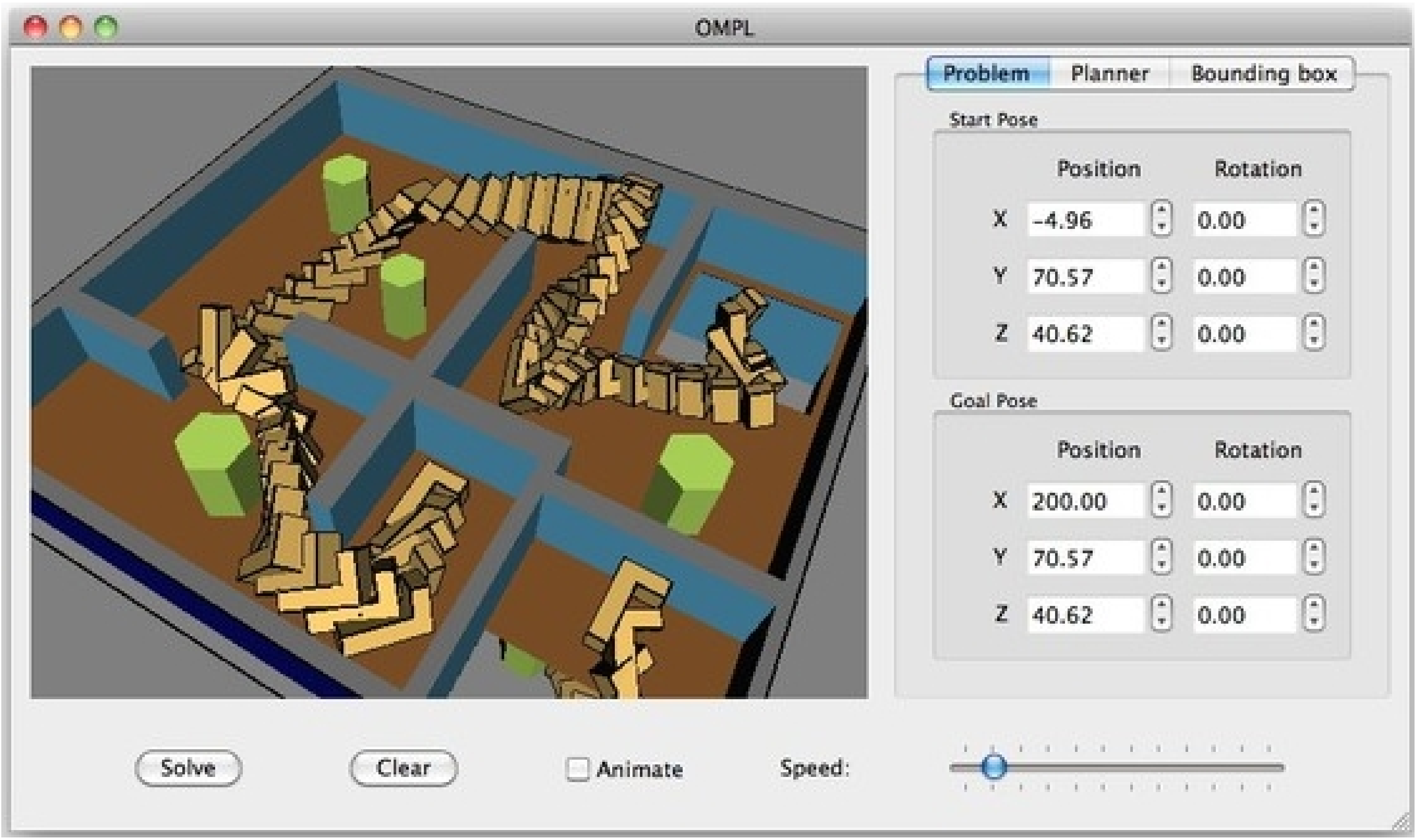


Figure from
http://www.gamasutra.com/blogs/MattKlingensmith/
20130907/199787/Overview_of_Motion_Planning.php

Open Motion Planning Library:
http://ompl.kavrakilab.org

The Pilot

- Navigation utility defined in cozmo_fsm/pilot.fsm
- How to go from A to B:
  - Generate obstacle list from current world map.
  - Use RRT-Connect to plan a path from A to B.
  - Formulate a navigation plan to follow the path.
    - Straight segments
    - Turns
    - Arcs
    - Landmark checks
  - Execute the navigation plan, correcting as necessary.
  - Report success or failure.

PilotToPose Node

- State node for invoking the Pilot.
- Tell it where you want to go, and (optionally) the desired heading at the destination.
- Use a heading of NaN if you don't care.
- =PILOT=> transition can check for success, errors.

```
go: PilotToPose(Pose(500, 0, 0, angle_z=degrees(90)))
go =PILOT=> Say(“Success”)
go =PILOT(StartCollides)=> Say(“Start collides!”)
```

Path Viewer

```
PilotToPose(Pose(300, 0, 0, angle_z=degrees(90)))
```