COMP3702/COMP7702
Artificial Intelligence

Module 1: Search — Part 3

Dr Archie Chapman

Semester 2, 2020

The University of Queensland
School of Information Technology and Electrical Engineering

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati

Week 4: Logistics

- **Assignment 1 is due in one week (Sept 4)** (10%)
- Please read and abide the code source referencing requirement
- Tutorials 2 and 3 will help you with Assignment 1, see the worked solutions and videos if you missed them
- An extra short video on A* search is on Blackboard
- **RiPPLE round 1 is also due in one week (Sept 4)** (2.5%)
- Assignment 2 will be released straight after Assignment 1 is due

Week 4 — Overview

Before: Discrete search

- State graph representation
- General structure of search algorithms
- Uninformed (blind) search (e.g. DFS, BFS, IDDFS, UCS)
- Informed search (GBFS, A* search)

Today: Search in continuous spaces mainly for robot motion planning

- Simplest case: Point robot in a 2D world
- Formulating the problem as a search problem and solving it
- More general case: e.g. Articulated robots: Formulating the problem
- Solving the problem using a Probabilistic Roadmap (PRM)

Table of contents

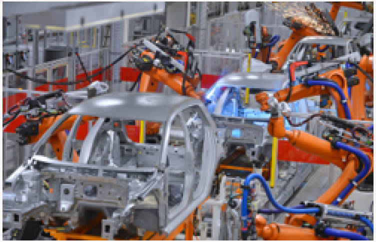
Introduction to continuous search for motion planning

Motion Planning

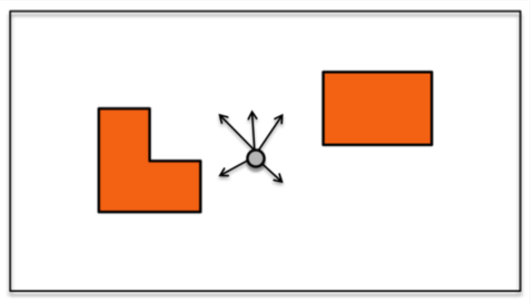
Motion planning is the study of computational methods to enable an agent to compute its own motions for moving from a given initial state to a goal state.

Various applications:

- Solving puzzles
- Assembling cars and planes
- Computer games
- Assignment 2
- Projects in METR4202



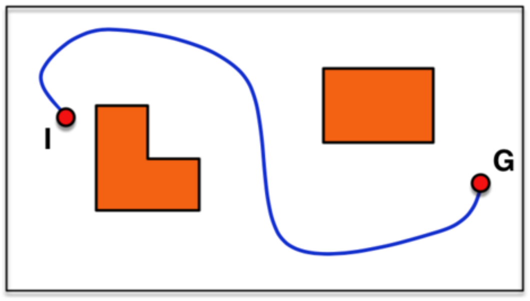
Simplest Motion Planning Problem: Point robot



Various applications:

- A point robot operating in a 2-dimensional workspace with obstacles of known shape and position
- We'll use "robot" in general terms.
- Loosely speaking, a robot is an agent that operates in the physical world or its simulation

Simplest Motion Planning Problem: Point robot



Find a collision-free path between a start and goal position of the robot.

Robot occupies a "point."

How to solve such a problem?

Essentially a search problem

How to solve such a problem?

Essentially a search problem ...but with continuous search and action space

How to solve such a problem?

Essentially a search problem . . . but with continuous search and action space

Size of state graph?

How to solve such a problem?

Essentially a search problem	...but with continuous search and action space
Size of state graph?	Naively, the size of the state graph is infinite!

How to solve such a problem?

Essentially a search problem . . . but with continuous search and action space
Size of state graph? Naively, the size of the state graph is infinite!
Key:

How to solve such a problem?

Essentially a search problem	. . . but with continuous search and action space
Size of state graph?	Naively, the size of the state graph is infinite!
Key:	Discretise to construct a more compact state graph

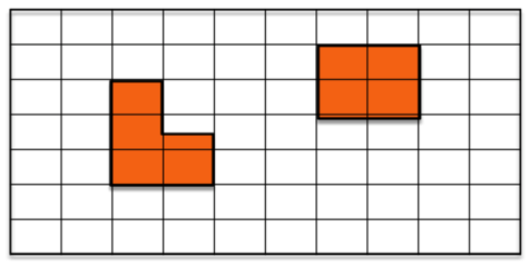
How to solve such a problem?

Essentially a search problem	...but with continuous search and action space
Size of state graph?	Naively, the size of the state graph is infinite!
Key:	Discretise to construct a more compact state graph
Many ways, e.g.:	

How to solve such a problem?

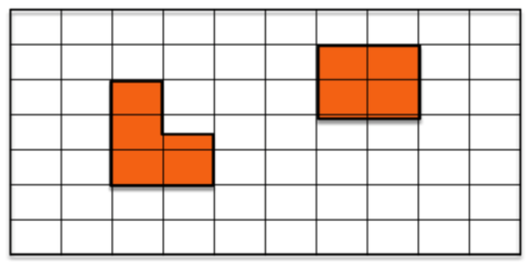
Essentially a search problem	. . . but with continuous search and action space
Size of state graph?	Naively, the size of the state graph is infinite!
Key:	Discretise to construct a more compact state graph
Many ways, e.g.:	Uniform grid discretization, Visibility graph

Uniform grid discretisation



- Obstacles do not have to be represented as polygons
- Each grid cell that does not intersect with an obstacle becomes a vertex in the state graph.
- Edges between vertex v and v' mean v and v' are neighbouring grid cells
- **Use search on state graph as usual** e.g. A* search.

Uniform grid discretisation



- Obstacles do not have to be represented as polygons
- Each grid cell that does not intersect with an obstacle becomes a vertex in the state graph.
- Edges between vertex v and v' mean v and v' are neighbouring grid cells
- **Use search on state graph as usual** e.g. A* search.

So we're done here. . . ?

Big problems

- When the dimensionality of the state space increases, the complexity (time and space) grows exponentially
- The dimension of the state space can be very large!

Question: what is the dimension of the state space of a UAV drone?

Visibility graph

Visibility graph (Shakey Project 1969)

Obstacles are represented as polygons

State space is a undirected graph where:

- Nodes are vertices of the obstacles
- An edge between two vertices represents an edge of the polygon or a collision-free straight line path between two vertices

Given an initial (I) and goal (G) states,

- Find the vertex q_i nearest to I, where the line segment between I and q_i is collision free
- Similarly for G

Visibility graph (Shakey Project 1969)

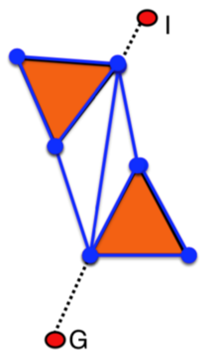
Obstacles are represented as polygons

State space is a undirected graph where:

- Nodes are vertices of the obstacles
- An edge between two vertices represents an edge of the polygon or a collision-free straight line path between two vertices

Given an initial (I) and goal (G) states,

- Find the vertex q_i nearest to I, where the line segment between I and q_i is collision free
- Similarly for G



Visibility graph

Algorithm 1 Visibility Graph Construction

```
1:  Input: q_I, q_G, polygonal obstacles
2:  Output: visibility graph G
3:  for every pair of nodes u, v do                      ▷ O(n^2)
4:      if segment(u, v) is an obstacle edge then         ▷ O(n)
5:          insert edge(u, v) into G
6:      else
7:          for every obstacle edge e do                  ▷ O(n)
8:              if segment(u, v) intersects e then
9:                  break and go to line 3
10:         insert edge(u, v) into G
```

If each edge is labelled with the length of the path the edge represents, the shortest path can be found by finding the shortest path in the graph

Complexity (n is total #vertices of the obstacles):

- Construction time (naïve): $O(n^3)$
- Space: $O(n^2)$

Visibility graphs have been extended (for efficiency) and are used quite a lot in games

Visibility graph

Pros:

- Independent of the size of the environment
- Can make multiple shortest path queries for the same graph (i.e. the environment remains the same, but the initial and goal states change)

Visibility graph

Pros:

- Independent of the size of the environment
- Can make multiple shortest path queries for the same graph (i.e. the environment remains the same, but the initial and goal states change)

Cons:

- Shortest paths always graze the obstacles
- Hard to deal with non-polygonal obstacles
- When the dimensionality of the state space increases (i.e. obstacles with many dimensions or lots of obstacles), the complexity (time and space) grows exponentially
- The dimension of the state space can be very large!

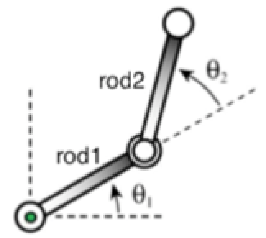
State space and configuration space (C-space)

More general case: Articulated Robot

Articulated robot operating in a 2D environment

- Articulated robot: A robot that consists of multiple rigid bodies, connected by joints.
- A rigid body: An object where the distance between any 2 points on the object remains the same when the object moves.

Let's start with a simple case: 2 rigid rods + 2 joints.



State space and C-space

State space: The set of all configurations

Configuration space / C-Space: is the set of all possible robot configurations

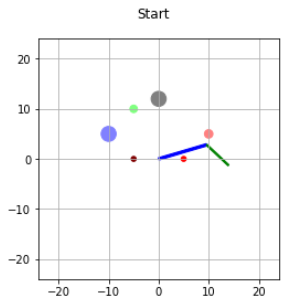
- A configuration is the parameters that uniquely define the position of every point on the robot: $q = (q_1, q_2, \ldots q_n)$
- Usually expressed as a vector of the Degrees of Freedom (DOF) of the robot
- **Motion planning**: Find a collision free path in this state space

State space and C-space

State space: The set of all configurations

Configuration space / C-Space: is the set of all possible robot configurations

- A configuration is the parameters that uniquely define the position of every point on the robot: $q = (q_1, q_2, \ldots q_n)$
- Usually expressed as a vector of the Degrees of Freedom (DOF) of the robot
- **Motion planning**: Find a collision free path in this state space

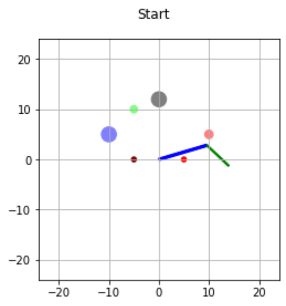


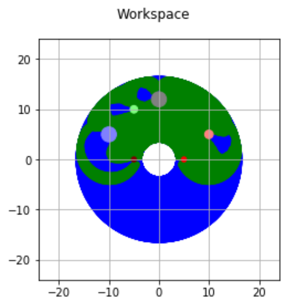
State space and C-space

State space: The set of all configurations

Configuration space / C-Space: is the set of all possible robot configurations

- A configuration is the parameters that uniquely define the position of every point on the robot: $q = (q_1, q_2, \ldots q_n)$
- Usually expressed as a vector of the Degrees of Freedom (DOF) of the robot
- **Motion planning**: Find a collision free path in this state space



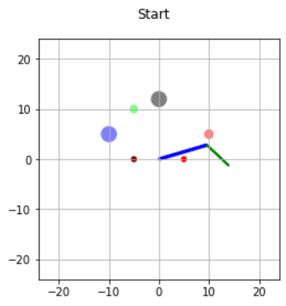


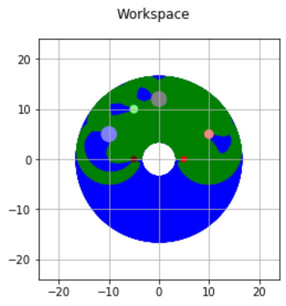
State space and C-space

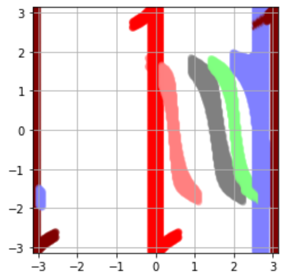
State space: The set of all configurations

Configuration space / C-Space: is the set of all possible robot configurations

- A configuration is the parameters that uniquely define the position of every point on the robot: $q = (q_1, q_2, \ldots q_n)$
- Usually expressed as a vector of the Degrees of Freedom (DOF) of the robot
- **Motion planning**: Find a collision free path in this state space







Some terminology in C-space

A configuration q is **collision-free** if the robot placed at q does not intersect any obstacles in the workspace

Forbidden region

- The set of configurations that will cause the robot to collide with the obstacles in the environment

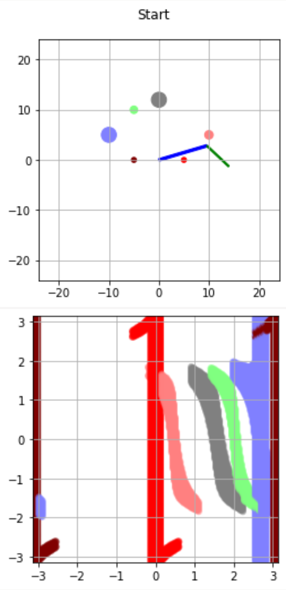
Free Space

- C-space forbidden region

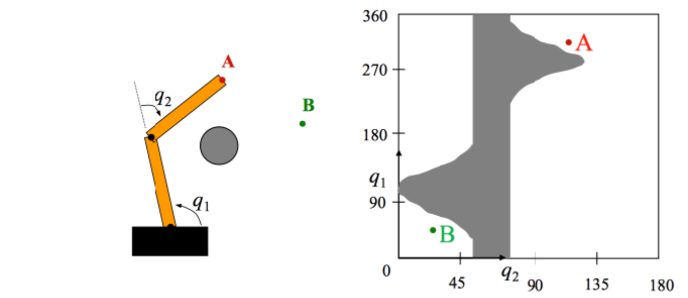
An **action**: a displacement vector

Transition: $q' = q + a$,

- where q and q' are configurations and a is an action



How do we get from A to B?

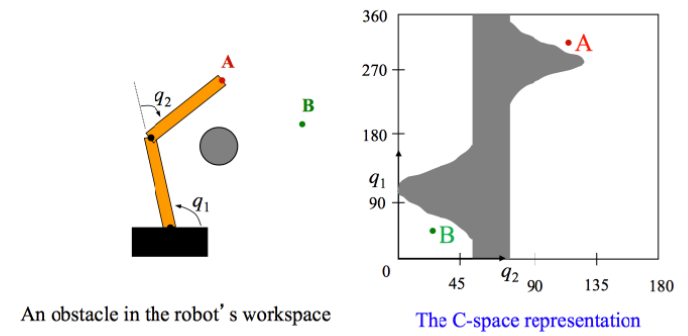


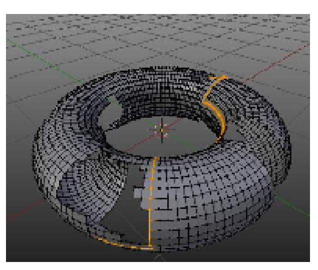
An obstacle in the robot's workspace

The C-space representation

Configuration space

How do we get from A to B?



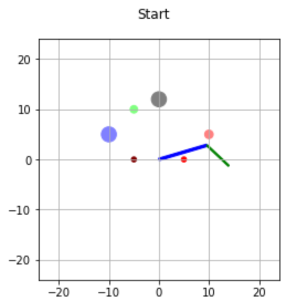


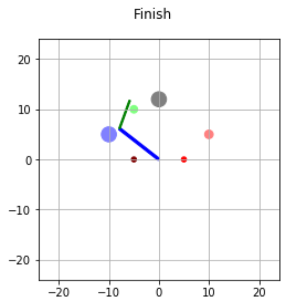
Using the same method as for the point robot

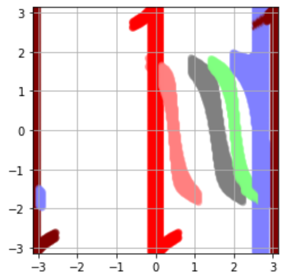
Visibility graph: Polygon?

Uniform grid discretization:

- Each grid cell that does not intersect with an obstacle becomes a vertex in the state graph
- Edges between vertex v and v' means v and v' correspond to neighbouring grid cells







Problem: #vertices is exponential in #dimensions.

Problem with uniform grid discretisation

As #joints increases, dimensionality of state space increases.

#grid cells in uniform grid discretization (i.e., #vertices in state graph) grows exponentially with #dimension of the state space.

- Should not store this state graph explicitly.

#out-edges grows exponentially with #dimension of the state space.

- Complexity of search algorithms depend on #out-edges in the state graph.

Problem with uniform grid discretisation

As #joints increases, dimensionality of state space increases.

#grid cells in uniform grid discretization (i.e., #vertices in state graph) grows exponentially with #dimension of the state space.

- Should not store this state graph explicitly.

#out-edges grows exponentially with #dimension of the state space.

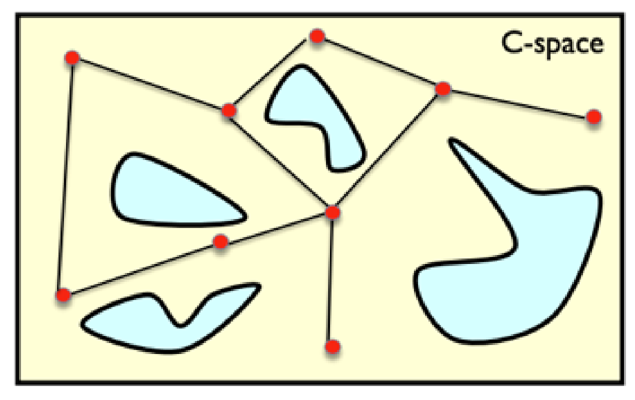
- Complexity of search algorithms depend on #out-edges in the state graph.
  Curse of dimensionality!

A better alternative. . .

Build a small state graph that captures only the “important features ” of the state space

- For motion planning, important features: Connectivity of free space

How? Use sampling to build the graph



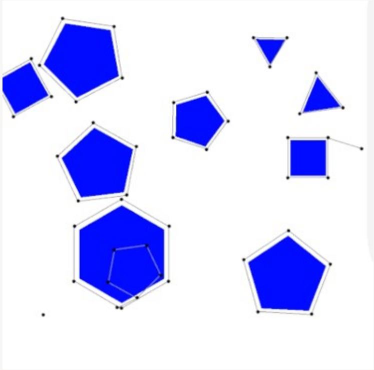
Probabilistic Roadmap (PRM)

Probabilistic Roadmap (PRM)

Kavraki, et al. '96

Sample a set of states uniformly at random

- Vertices in the state graph (called roadmap)

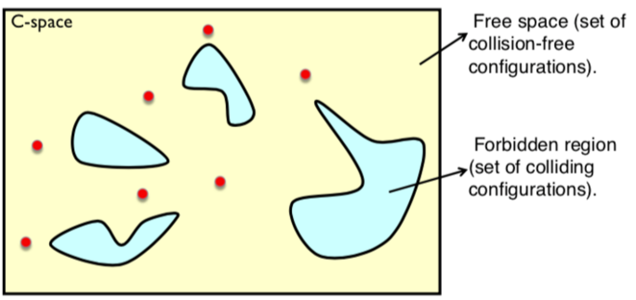


Probabilistic Roadmap (PRM)

Kavraki, et al. '96

Sample a set of states uniformly at random

- Vertices in the state graph (called roadmap)

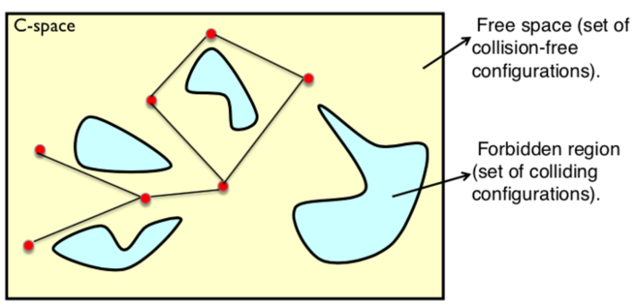


Probabilistic Roadmap (PRM)

Kavraki, et al. '96

Sample a set of states uniformly at random

- Vertices in the state graph (called roadmap)

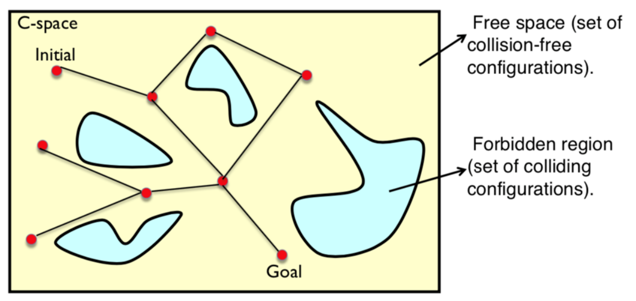


Probabilistic Roadmap (PRM)

Kavraki, et al. '96

Sample a set of states uniformly at random

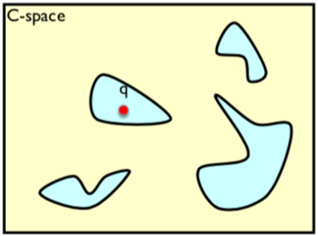
- Vertices in the state graph (called roadmap)



State graph / Roadmap construction

Loop:

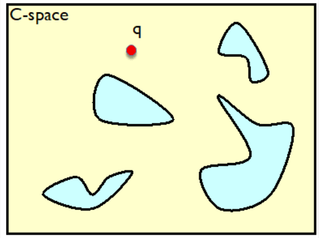
- Sample a configuration q uniformly at random from the state space
- If q is not in-collision,
    - Add q as a vertex to the state graph



State graph / Roadmap construction

Loop:

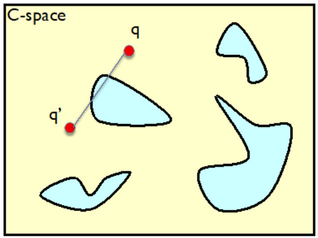
- Sample a configuration q uniformly at random from the state space
- If q is not in-collision,
  - Add q as a vertex to the state graph



State graph / Roadmap construction

Loop:

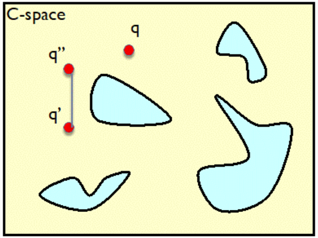
- Sample a configuration q uniformly at random from the state space
- If q is not in-collision,
  - Add q as a vertex to the state graph
  - For all vertices q' within D distance from q in state graph:
  - If the line segment (in C-space) between q and q' is not in-collision, add an edge qq' to the state graph



State graph / Roadmap construction

Loop:

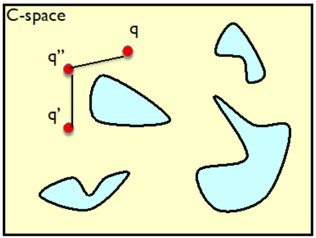
- Sample a configuration q uniformly at random from the state space
- If q is not in-collision,
    - Add q as a vertex to the state graph
    - For all vertices q' within D distance from q in state graph:
    - If the line segment (in C-space) between q and q' is not in-collision, add an edge qq' to the state graph



State graph / Roadmap construction

Loop:

- Sample a configuration q uniformly at random from the state space
- If q is not in-collision,
    - Add q as a vertex to the state graph
    - For all vertices q' within D distance from q in state graph:
    - If the line segment (in C-space) between q and q' is not in-collision, add an edge qq' to the state graph



Once the state graph is constructed

Given an initial and a goal configuration,

- Find the vertex q_i nearest to the initial configuration, where the straight line segment between initial configuration and q_i is collision free
- Find the vertex q_i nearest to the goal configuration, where the straight line segment between goal configuration and q_i is collision free
- Find a path from q_i to q_i using the search algorithms we discussed (blind/informed search)

But when is the state graph construction “done ”?

- i.e. what’s the stopping criteria? time limit, or . . . ?

Interleave

Interleave state graph construction *and* graph search

Interleave

Interleave state graph construction *and* graph search... until all queries are answered, i.e., found a path from each pair of initial and goal configurations.

while runtime $<$ timeLimit **AND** path is not found

repeat:

Sample a configuration q with a suitable sampling strategy
if q is collision-free **then**:

Add q to the graph G
Connect q to existing vertices in G using valid edges

until n new vertices have been added to G

Search G for a path

Probabilistic Roadmap (Summary)

State space $\rightarrow$ C-space $\rightarrow$ generate search graph $\rightarrow$ search on a graph

Use sampling to construct the state graph.

Key components:

1. Sampling strategy (adding vertices)
2. Connection strategy (adding edges)
3. Check if a configuration is valid or not
4. Check if a line segment in C-space is valid or not

Overview of PRM

1. **Sampling strategy (adding vertices)**
2. Check if a configuration is valid or not
3. Connection strategy (adding edges)
4. Check if a line segment in C-space is valid or not

PRM – State Graph (Roadmap) construction

repeat:

Sample a configuration q with a suitable sampling strategy

if q is collision-free **then**:

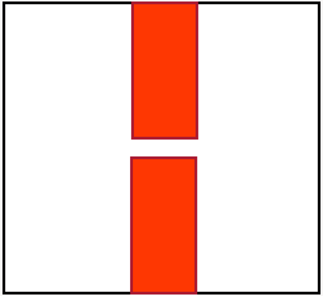
Add q to the graph G

Connect q to existing vertices in G using valid edges

until n new vertices have been added to G

Problems?

Original PRM uses uniform distribution

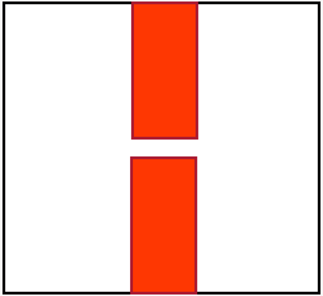


Theory: probability of a point in any $\epsilon - ball$ being sampled at a given iteration is always > 0

$$\Rightarrow \lim_{\#samples \to \infty} [\text{probability of a feasible path being sampled}] = 1$$

Problems?

Original PRM uses uniform distribution



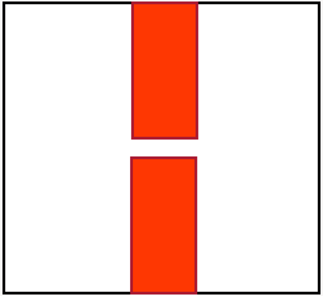
Theory: probability of a point in any $\epsilon - \mathit{ball}$ being sampled at a given iteration is always > 0

$$\Rightarrow \lim_{\#\mathit{samples} \to \infty} [\text{probability of a feasible path being sampled}] = 1$$

Can we do better than this?

Problems?

Original PRM uses uniform distribution

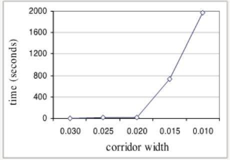


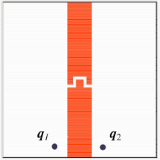
Theory: probability of a point in any $\epsilon - ball$ being sampled at a given iteration is always > 0

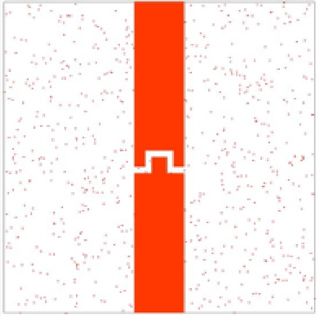
$$\Rightarrow \lim_{\#samples \to \infty} [\text{probability of a feasible path being sampled}] = 1$$

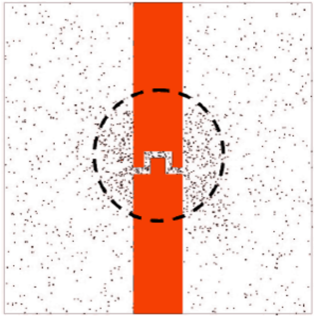
Can we do better than this? Yes we can, using heuristic **sampling strategies**

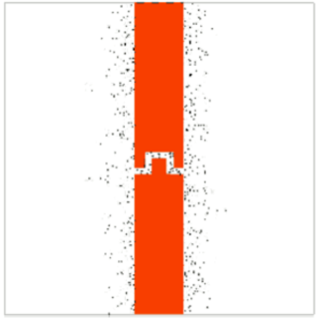
PRM – State Graph (Roadmap) construction









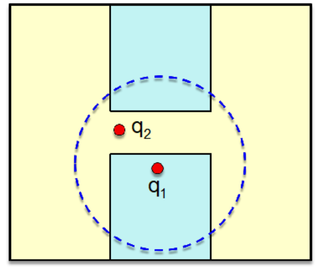


Sampling strategy 1 – Sample near obstacles

1. Sample a configuration q_1 uniformly at random.
2. Sample a configuration q_2 from the set of all configurations within D distance from q_1, uniformly at random.
3. If q_1 is in-collision and q_2 is collision-free

   Add q_2 as a vertex in the state graph
4. Else if q_1 is collision-free and q_2 is in-collision

   Add q_1 as a vertex in the state graph.

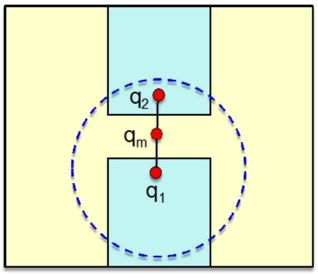


Sampling strategy 2 – Sample inside a passage

1. Sample a configuration q_1 uniformly at random.
2. Sample a configuration q_2 from the set of all configurations within D distance from q_1, uniformly at random.
3. If q_1 and q_2 are in-collision,

   Check if the middle configuration $q_m = 0.5 * (q_1 + q_2)$ is collision free.

   If q_m is collision-free, add q_m as a vertex in the state graph.



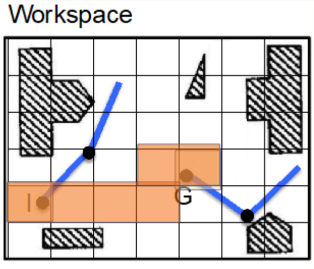
Sampling strategy 3 – Using workspace information

Narrow passages in C-space are often caused by narrow passages in the workspace.

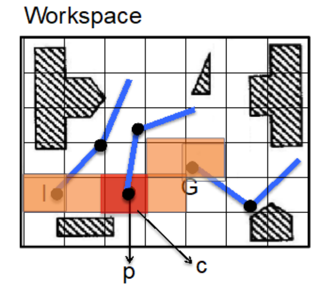
Relax problem into planning for a point robot, break task into two phases, in w-space then c-space.

First:

1. Discretize the workspace into uniform grid.
2. Choose a point r on the robot.
3. Find a path π assuming the robot is the point r.
   - π: sequence of grid cells.



Sampling strategy 3 – Using workspace information



Second, to sample a configuration:

1. Sample a cell c in π with equal probability.
2. Sample a point p uniformly at random from c.
3. Sample configurations uniformly at random from the set of all configurations that place point r of the robot at p.

Combining sampling strategies

Analogous to combining search heuristics in A* search

- A sampling strategy = a heuristic
- Many powerful heuristics
- Key in Watson

How to combine?

- Simplest:
    - Assign equal weight to each candidate sampling strategy
    - Choose a strategy to use randomly based on the weight
    - Use the chosen strategy to sample a configuration
    - Repeat the above steps, without changing the weight

Combining sampling strategies

Analogous to combining search heuristics in A* search

- A sampling strategy = a heuristic
- Many powerful heuristics
- Key in Watson

How to combine?

- Simplest:
  - Assign equal weight to each candidate sampling strategy
  - Choose a strategy to use randomly based on the weight
  - Use the chosen strategy to sample a configuration
  - Repeat the above steps, without changing the weight
- **Alternative:** Adapt the weights: frame as a multi-arm bandit problem

Multi-armed bandit problem

Each arm = a sampling strategy

Choose which "arm " would be most helpful in solving the problem

Key: **trading off exploration vs exploitation**

- Use the sampling strategy that has shown to be useful, but it may not be the best strategy (exploitation)
- Use other sampling strategy that may not have shown good performance yet, but may actually be the best strategy (exploration)

Note: We will cover MABs in detail later in the course.

Multi-arm bandit problem

Many solutions, many policies

Simplest: **epsilon-greedy**

Assign a weight to each sampling strategy

Start with equal weight for all strategies

Loop:

Strategy with the highest weight is selected with probability $(1 - \epsilon)$, or another is selected with probability ϵ/N, where N is #strategies available

Suppose strategy s_1 is selected; then we'll use s_1 to sample and add a vertex and edges to the roadmap

- If the addition connects disconnected components of the roadmap OR adds #connected components of the roadmap, increment the weight of s_1 by 1

Alternative policies use variance infoamtion: EXP3, UCB, KUBE...for later in the course

Overview of PRM

1. Sampling strategy (adding vertices)
2. **Check if a configuration is valid or not**
3. Connection strategy (adding edges)
4. Check if a line segment in C-space is valid or not

Are two axis-aligned rectangles colliding?

Axis-aligned rectangles: each side of the rectangle is parallel to the X axis or to the Y axis.

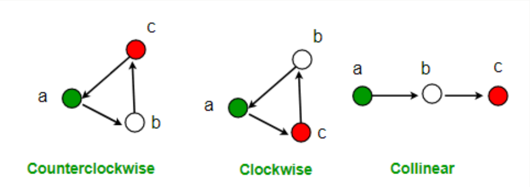
Suppose A and B are axis-aligned rectangles,

A and B do not collide when:

- The left edge of A is on the right side of the right edge of B
- The right edge of A is on the left side of the left edge of B
- The top edge of A is below the bottom edge of B
- The bottom edge of A is above the top edge of B

If A and B do not satisfy at least one of the four points above, they collide

How about other polygons?



Orientation of a sequence of 3 points a, b, c, based on determinant of cross product:

$$\text{Area}(a, b, c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$$

Orientation of abc test (CCW):

Counter-clockwise: $\text{Area}(a, b, c) > 0$

Clockwise: $\text{Area}(a, b, c) < 0$

Colinear: $\text{Area}(a, b, c) = 0$

Simple Collision Check: Line Segment – Line Segment

Suppose we have 2 line segments: $p_1 q_1$ and $p_2 q_2$. The line-segments intersect whenever the :

(Majority of the cases:)

If $p_1 q_1 p_2$ and $p_1 q_1 q_2$ have different orientations

AND

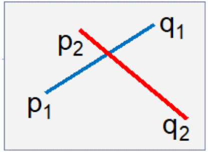
$p_2 q_2 p_1$ and $p_2 q_2 q_1$ have different orientations

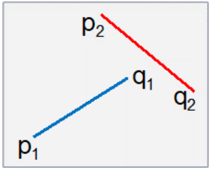
- Special cases: All co-linear segments will intersect when:

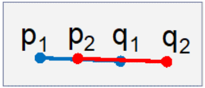
  The x components of the first segment intersects with that of the second segment

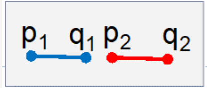
  AND

  The y components of the first segment intersects with that of the second segment









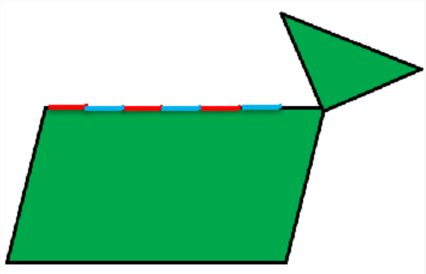
Simple Collision Check: Line Segment – Line Segment (more efficient)

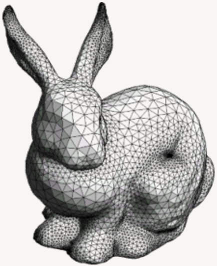
- Bound the line segments with axis-aligned rectangles, i.e. bounding boxes
- Check intersection between rectangles
  - Only need to check range of x and y.
- Only if the bounding rectangles intersect that we perform the collision check as in the previous slide.

A More General Collision Check: Hierarchical Bounding Volume

Suppose we have an object O, whose boundary is represented by geometric primitives, e.g.,

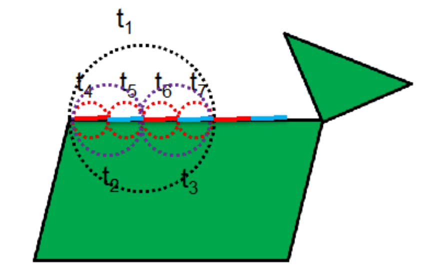
- The primitives of a polygon are short line segments
- The primitives of a polyhedron (rep. as triangular mesh) are triangles

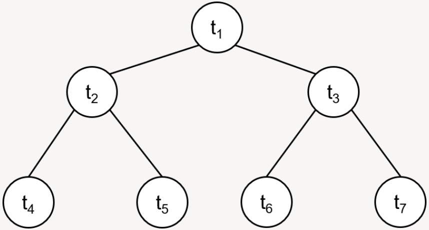




A More General Collision Check: Hierarchical Bounding Volume

Constructing a tree of bounding volumes for each object we want to check





Idea: Use bounding volumes where collision check is easy (e.g., sphere). If the bounding volumes are separated, definitely no collision

A More General Collision Check: Hierarchical Bounding Volume

How to construct a tree of bounding volumes for each object we want to check?

- Each node represents a bounding volume (sphere / box)
- The root covers the entire object. Each leaf node covers a geometric primitive of the object
- The higher the node (the closer to the root), the corresponding bounding volume covers a larger surface area of the object
- Construct bottom up

Use the tree of bounding volumes to quickly check collision between the objects

Where do we gain efficiency?

The hope is that we don't need to compute distance with all geometric primitives

- If we do, then we don't gain anything using this method.
- However, in most practical scenarios, these methods do eliminate a lot of distance computation between the point and geometric primitive of the object

Overview of PRM

1. Sampling strategy (adding vertices)
2. Check if a configuration is valid or not
3. **Connection strategy (adding edges)**
4. Check if a line segment in C-space is valid or not

Where should we try adding the edges?

All pairs?

- Long edges: expensive and have higher change of colliding with the forbidden region
- Try inserting edges only when the distance between two vertices are relatively small.
  - Too small: Require more samples to solve the problem
  - Too large: Similar problem as all pairs
  - Need a balance
  - Sometimes, we also limit the number of edges we try to add to each vertex. Or limit the degree of each vertex.
- Usually need some trial and error.

Where should we try adding the edges?

Lazy approach:

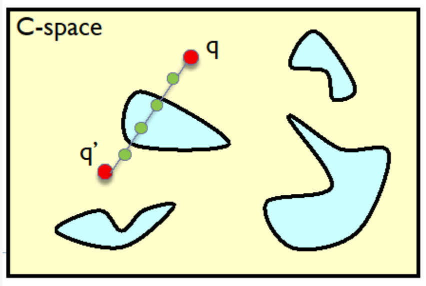
- Add edges to all pairs of vertices located within a certain distance of each other (no collision check)
- Do collision check only when needed, i.e., when we search for a path and the particular edge may be traversed.

Overview of PRM

1. Sampling strategy (adding vertices)
2. Check if a configuration is valid or not
3. Connection strategy (adding edges)
4. **Check if a line segment in C-space is valid or not**

Collision check for a line segment: Simple method

1. Discretize the line segment into small segments.
2. For each small segment,
    - Take the mid-point q_m to represent the small segment.
    - Check if q_m is in forbidden region or not (i.e., if a robot at configuration q_m collides with an obstacle).
    - If q_m is in forbidden region, return collision.

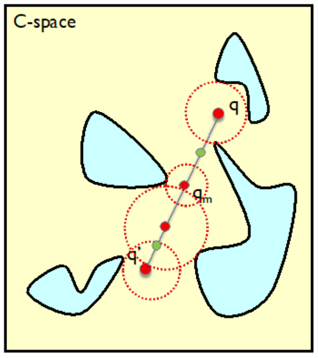


Collision check for a line segment: Simple method

Problems:

- Discretisation might miss a spot OR
- Might need to do extremely large number of collision checks

Collision check for a line segment: A better method using distance computation



1. If q or q' is in collision, return
2. Compute distance d_q between q and its nearest forbidden region. Create an empty ball B_q with centre q and radius d_q. Similarly for q'
3. Let $q_m = (q + q')/2$
4. If q_m is inside B_q and Bq', the entire segment qq' is collision free. Otherwise, repeat from #1, but for segments qq_m and $q_m q'$

Guarantees that entire path is collision free; Also more efficient

Probabilistic Roadmap (Summary)

State space $\rightarrow$ C-space $\rightarrow$ generate search graph $\rightarrow$ search on a graph Many variants of PRM, varying each of its key components (the four topics we just covered)

- PRM is a **sampling-based planner**
    - It uses sampling to construct the state graph
- Others **build search tree directly**:
    - Expansive Space Tree (EST)
    - Rapidly exploring Random Trees (RRT)

Attributions and References

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati for their materials.

Other material derived from *Principles of Robot Motion* (Choset, et al. 2005, MIT press), and slides by Howie Choset. See Blackboard for relevant excerpts from this book.

All remaining errors are Archie's — please email if you find any: archie.chapman@uq.edu.au