Computational Principles of Mobile Robotics

Planning in, representing and reasoning about space

7.1 Representing the robot

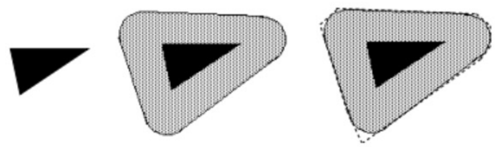
- Configuration space (C-space)
  - One degree of freedom for every degree of freedom of the robot
    - q=(x,y,θ) for a point robot.
    - q=($\theta_1, \theta_2, \dots, \theta_n$) for a robot with n joints.
  - Given the geometry of the robot and its configuration we can determine what part of space the robot ($A(q)$).
  - An obstacle prohibits certain configurations of the robot
    - $CB_i = \{q \in C | A(q) \cap B_i \neq \emptyset\}$.
  - Free space is the space that a robot intersects no obstacle
    - $= \{q \in \mathrm{C} | A(q) \cap (\cup_i B_i) = \emptyset\}$.

7.1.1 Configuration space

- Holonomic constraints
  - Can be written in the form G(q)=0 where q is the robot pose.
- Constraints on the derivatives of the robot pose q
  - $G\left(q, \frac{dq}{dt}, \frac{d^2q}{dt^2}, \ldots\right) = 0$ are non-holonomic.
- Robots whose motion is holonomic are much easier to plan paths for than robots with non-holonomic constraints
  - Parallel parking is the classic example.

7.1.2 Simplifications

- It is often very difficult (computationally difficult) to model all of the details of a given robot.
- A common solution is to dilate all objects and to shrink the robot to a point and to assume that the robot is capable of holonomic motion
  - This is the point robot assumption.
  - Quite similar to the block robot introduced earlier.

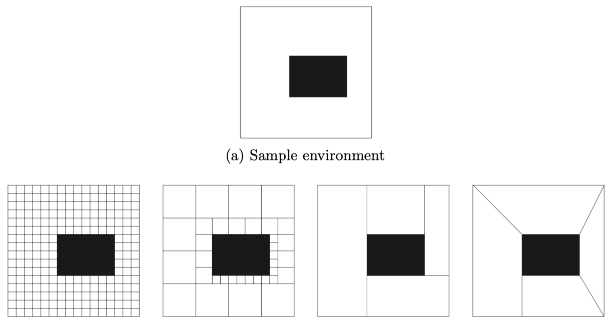


7.2 Representing space

- Although there are some things one can do without representing space, having some representation (a map) allows
  - To establish those parts of the environment that are free for navigation.
  - To recognize regions or locations in the environment.
  - To recognize specific objects in the environment.
- Note that not all tasks require a representation (a map)
  - Reactive mechanisms do not.
  - Certain policies do not (follow that line, keep your left hand on the wall, etc.).

7.2.1 Spatial decomposition

- Basic approach: Assume some N dimensional Cartesian space
- Sample it discretely.



(a) Sample environment

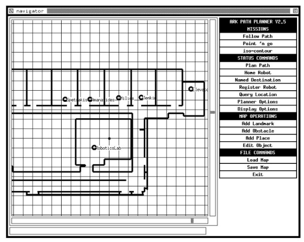
(b) Uniform (c) Quadtree (d) BSP (e) Exact

```
quadtree(Region) ::=
   condition = homogeneity_test(Region)
   if condition in {empty,full} then
     return leafNode(condition)
   else
     begin
       region1 = topleft(Region)
       region2 = topright(Region)
       region3 = bottomleft(Region)
       region4 = bottomright(Region)
       topleftSubtree = quadtree(region1)
       toprightSubtree = quadtree(region2)
       bottomleftSubtree = quadtree(region3)
       bottomrightSubtree = quadtree(region4)
       return treeNode(topleftSubtree,toprightSubtree,
                       bottomleftSubtree,bottomrightSubtree)
     end
```

Quadtree decomposition algorithm

7.2.2 Geometric representations

- Describe the environment as an embedding of some set of primitives.
  - Typically include composition/deformation operations on same
- 2D maps, as a collection of
  - Points
  - Lines
  - Circles
  - Polynomials
  - Polyhedra
  - Splines



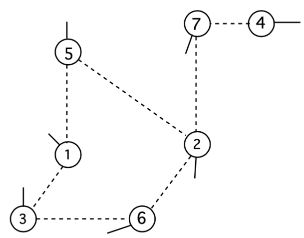
(a) Graphical view

```
landscape newworld;
  unit meter;
  width 22.500000 to 45.500000;
  height 12.000000 to 28.500000;
  elevation 0.000000 to 2.200000;
  begin
    name (43.541718,23.182617) "Elevator";
    name (33.433750,17.051640) "Robotics Lab";
    obstacle boundary2: wall;
      extent (39.849998, 25.200001) to (39.849998, 21.400000);
      elevation 0.000000 to 0.000000;
    end;
    obstacle boundary6: wall;
      extent (38.496956, 12.395000) to (38.496956, 12.065001);
      elevation 0.000000 to 0.000000;
    end;
end.
```

(b) Map definition language

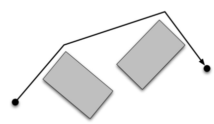
7.2.3 Topological representations

- Describe the world as an embedded graph

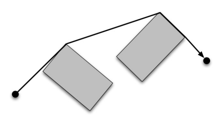


7.3 Path planning for mobile robots

- Given a start state (in configuration space) and a goal state (in configuration space) find an obstacle free continuous path from the start to the goal.
- Formally seek $\tau: s \in [0,1] \rightarrow C_{free}$ with $\tau(0) = q_{start}$ and $\tau(1) = q_{goal}$
- Free path lies completely within the free space.
- A semi-free path may touch the boundary of free space.



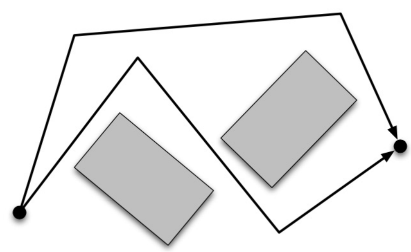
(a) Free-space path



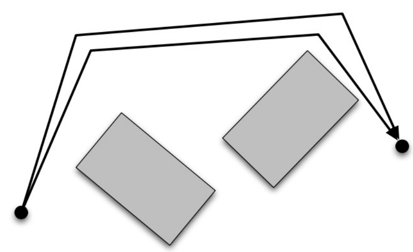
(b) Semi-free-space path

Path planning for mobile robots

- Homotopic classes



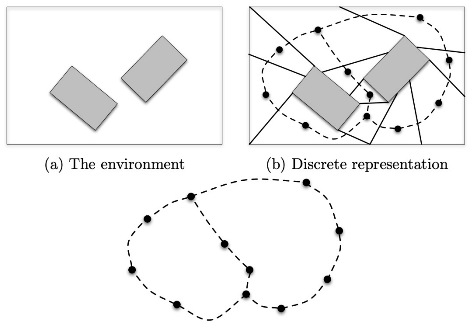
(a) Non-homotopic paths



(b) Homotopic paths

7.3.1 Constructing a discrete search space

- Take space (continuous) and sample it in some discrete manner and execute the process of finding a path in this discrete space.

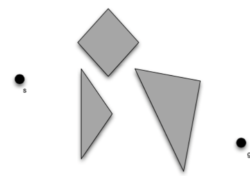


(a) The environment (b) Discrete representation

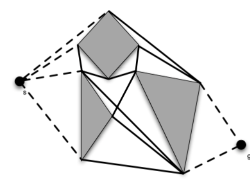
(c) Graph-based representation

```
Procedure VisibilityGraph
VG := ({n_i}, ∅).
n_i is any vertex on obstacles plus start and goal
for every pair of nodes u and v in VG do
    If e = (u, v) is an obstacle edge then
        add e to VG
    else
        for every obstacle edge o
            if e intersects o
                continue
        insert e in VG
end for
```

(a) Algorithm



(b) Graph



(c) Visibility graph

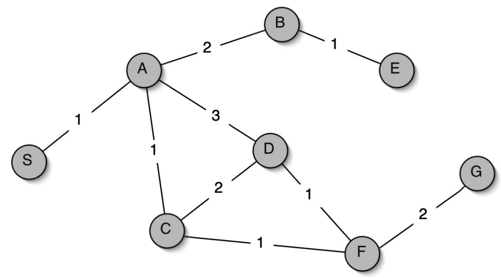
7.3.2 Retraction methods

- Reduce the dimensionality of space (to a set of edges in 2d).
- Plan paths on these edges and their intersection
  - Roadmap method.
- Voronoi graph is perhaps the best known.

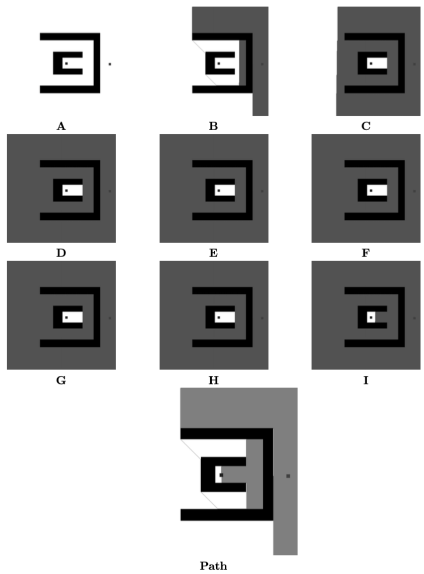
7.3.3 Searching a discrete search space

- Continuous space is reduced to a discrete set of states with connections between them.
- Call the set of states Nodes, and the connections edges
  - Apply classic graph search algorithm to solve the path planning problem

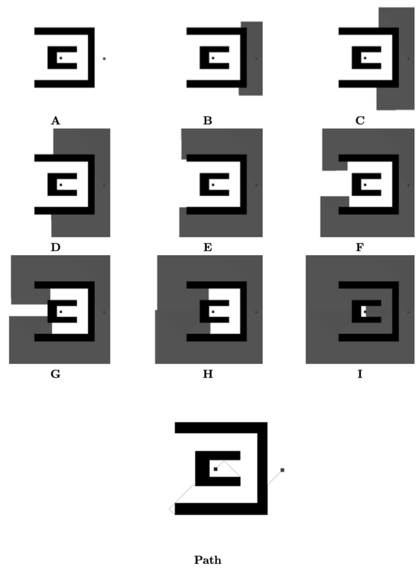
```
Procedure GraphSearch(s, goal)
OPEN := {s}.
CLOSED := {}.
found := false.
while (OPEN ≠ ∅) and (not found) do
    Select a node n from OPEN.
    OPEN := OPEN − {n}.
    CLOSED := CLOSED ∪ {n}.
    if n ∈ goal then
        found := true.
    else
        begin
            Let M be the set of all nodes
            directly accessible from n
            which are not in CLOSED.
            OPEN := OPEN ∪ M.
        end
end while
```



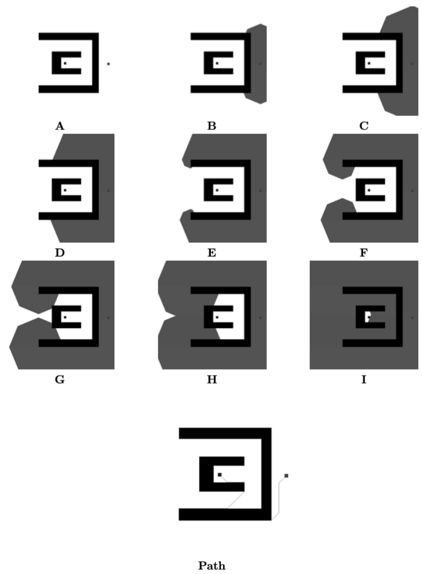
Depth first search



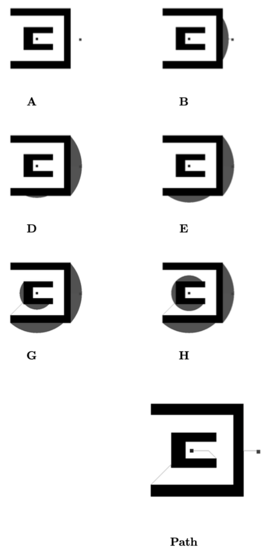
Breadth first search



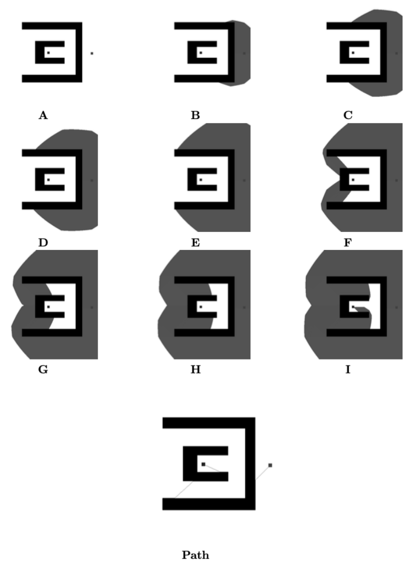
Dijkstra's algorithm



Best First search



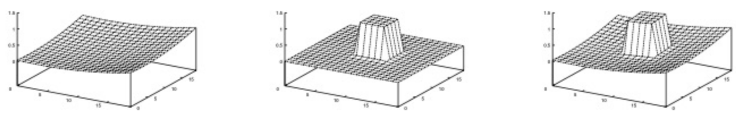
A*



Potential field methods

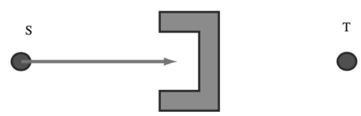
- Approach original proposed for real-time collision avoidance [Khatib, 86]
- Potential field: Scalar function over free space
- Ideal field (navigation function): smooth, global minimum at the goal, no local minima, grows to infinity near obstacles.
- Force applied to robot: negated gradient of the potential field. Always move along that force

Potential field methods



(a) Goal field (b) Obstacle field (c) Sum

Potential field methods



Robot trap

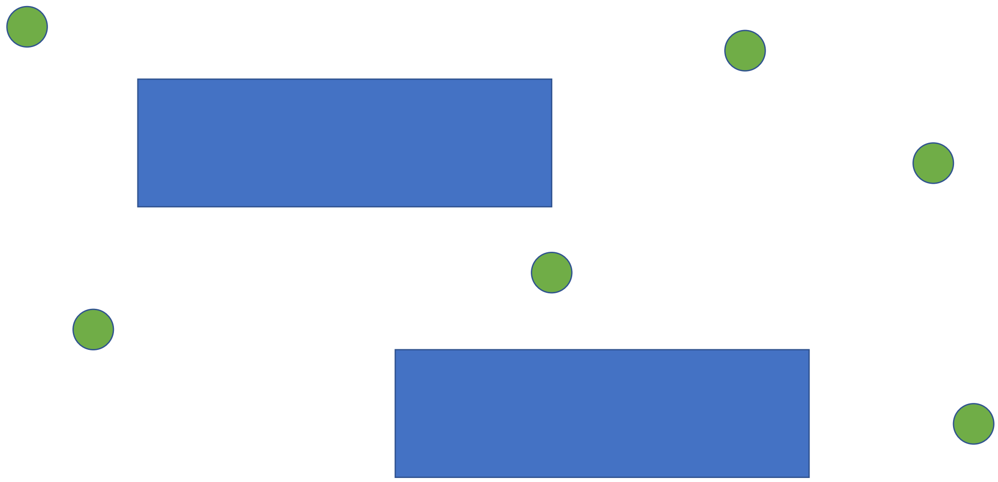
7.3.4 Spatial uncertainty

- Planning in the presence of uncertainty is difficult.
- A common assumption of the algorithms presented is that we know some things (environment model, robot state) which may not be well known.
- Performance of classic planning algorithms under such uncertainty is not well known.

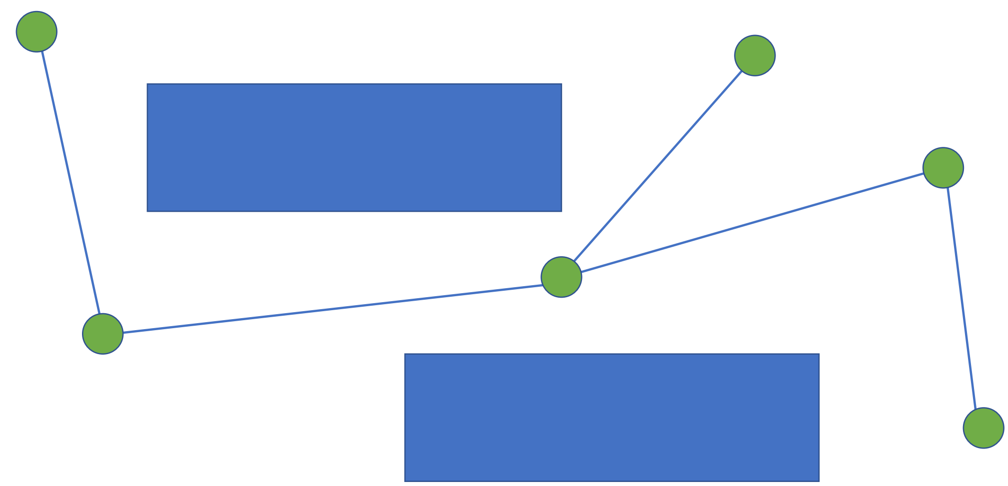
7.3.5 Complex environments

- Planning is exponential in DOF of the problem
  - As the dimensionality increases it becomes impractical to use classical planning techniques to solve the problem.
- Probabilistic solutions become necessary.
- Classic solution here is known as probabilistic path planning
  - RRT, RRT* are well known approaches here.

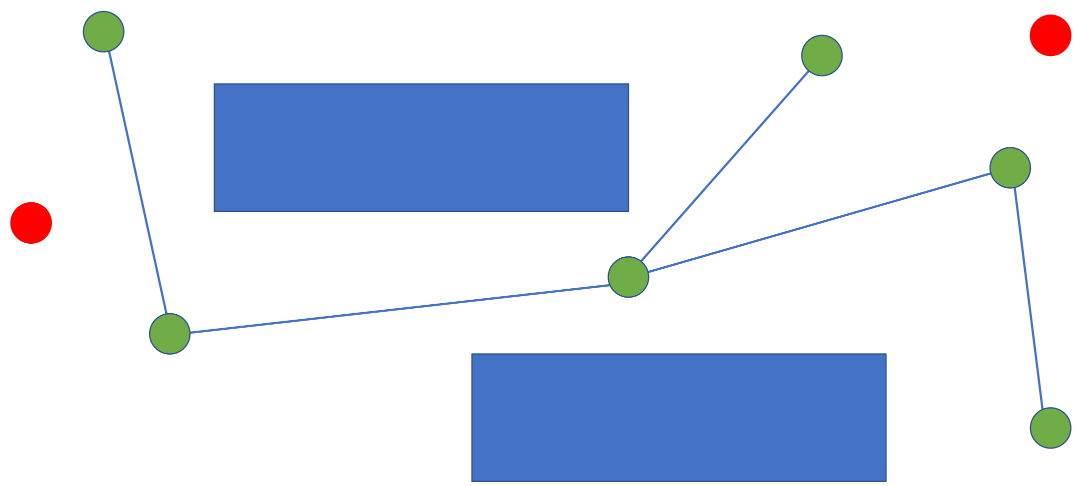
The Environment



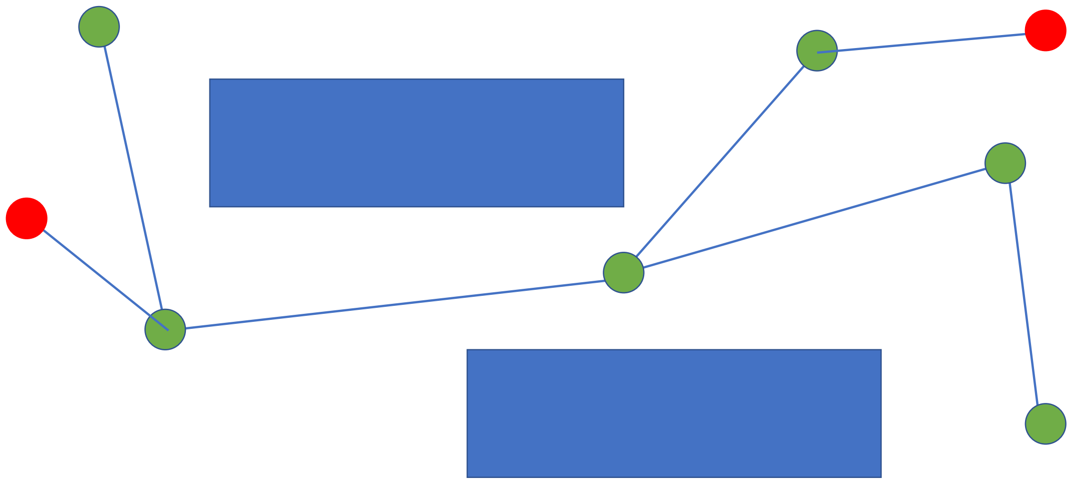
Randomly generate seed points



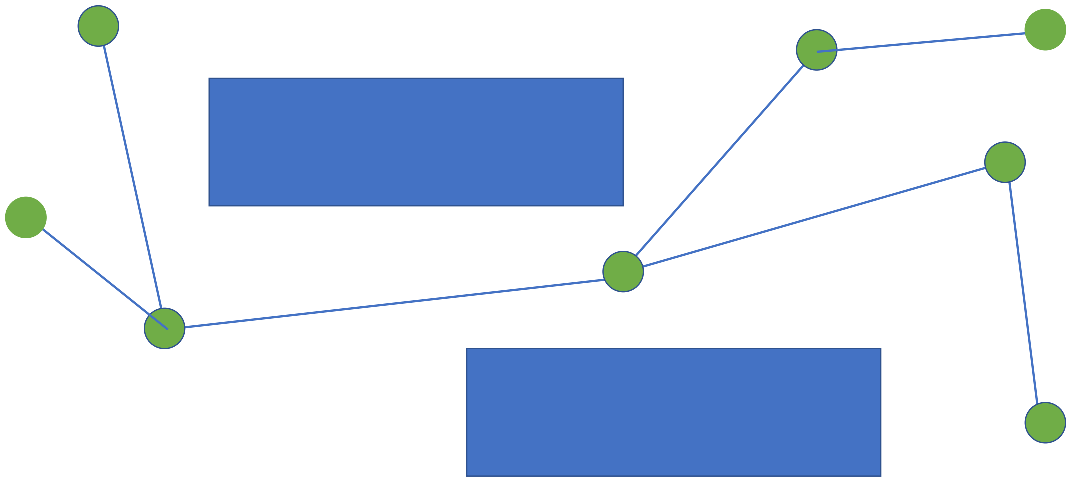
Construct a roadmap



Process query



Process query: find path



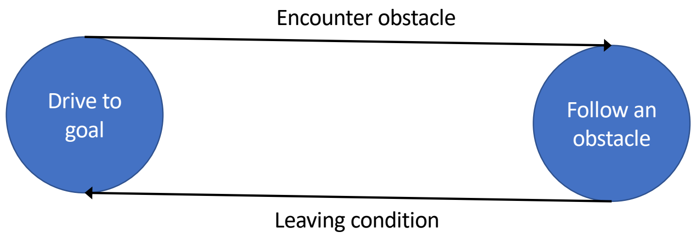
Enlarge graph

Complex environments

- RRT (Rapidly exploring random tree) and RRT* are best known.
  - Generate trees rather than graphs.
- RRT* as number of nodes approaches infinity, RRT* will deliver the shortest possible path to the goal.

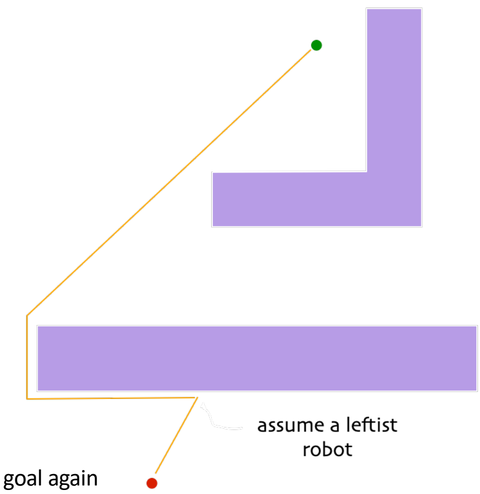
Complex environments

- Planning without a map
  - Bug algorithms.



Bug algorithms

- Known direction to goal
- Otherwise only local sensing (walls/obstacles, encoders)
- We first saw these in Chapter 2
  - Large literature on the approach



Bug 0
1) Head towards goal
2) Follow obstacle until you can head towards the goal again

Why Bug algorithms?

- No map of the environment.
- Can be extended to local sensor models
  - Vis-bug, tangent-bug
- Interesting practical and theoretical results

RL solutions

- More recently considerable success in developing RL-based approaches for both known and novel environments.
- Basic approach is to build a policy that 'is likely to work' to find a way from the start to the goal.

7.4 Planning for multiple robots

- Planning for multiple robots can be centrally controlled or distributed.
  - Central control -> basically increases the dimensionality of the problem.
  - Distributed control -> introduces a range of problems related to distributed algorithms and their execution.

7.5 Biological mapping

- Many biological agents have some internal representation of space and ability to plan and execute long-duration plans.
  - Migratory birds as an example.
- In animals, the brain structure known as the hippocampus is through to be critical to the construction of spatial maps.