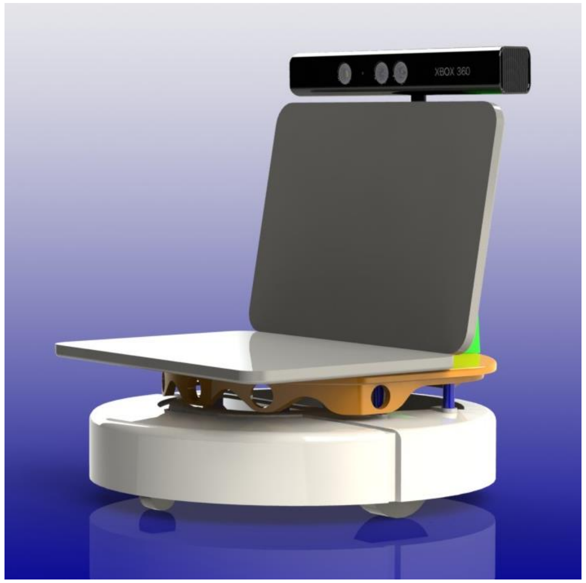
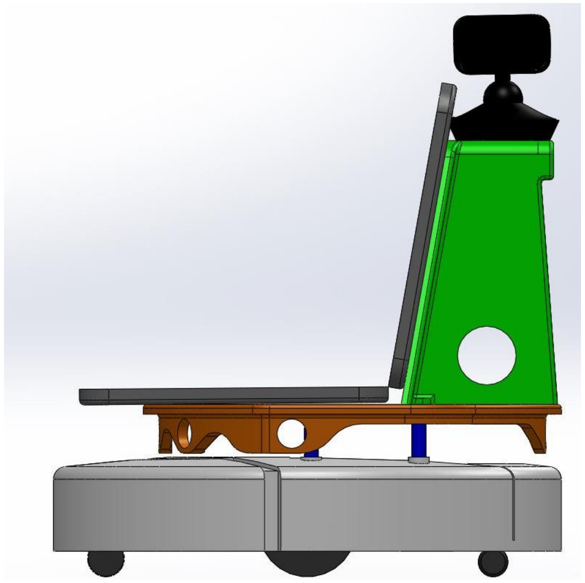
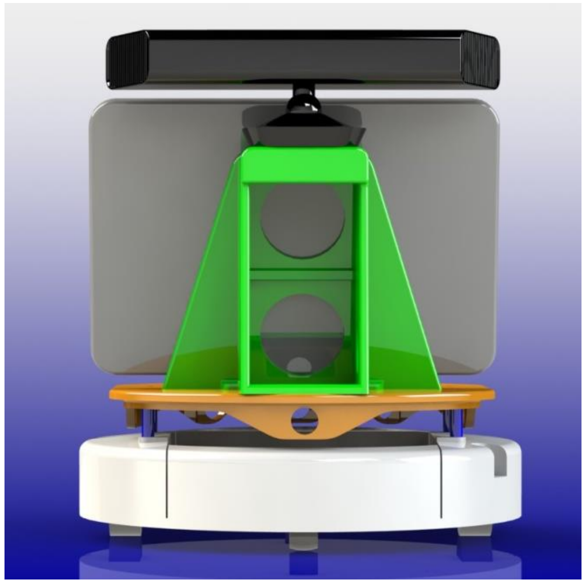
MAE 493G, CpE 493M, Mobile Robotics

11. Introduction to Robot Planning

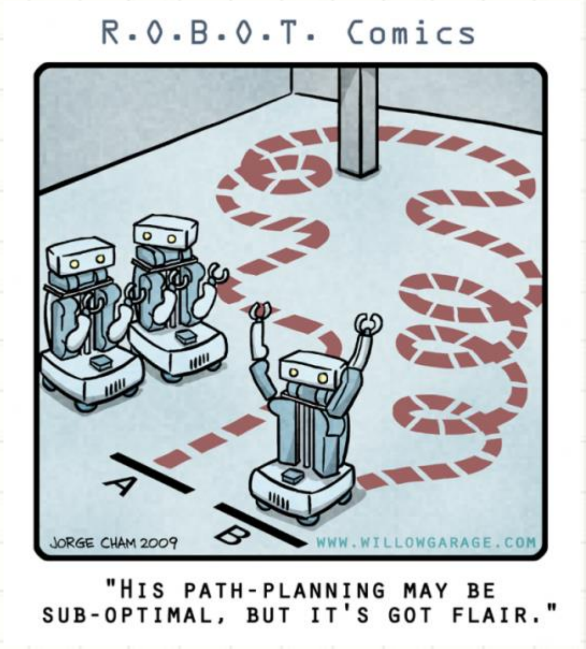


Instructor: Yu Gu, Fall 2013

Robot Planning

- A robot needs to be able to plan its motion to show intelligent behaviors;
- The plan can be based on long-term memory (e.g. models), short-term memory, both, or neither...
- Some planners use global knowledge (e.g. a map) while others uses only local knowledge (e.g. sensor readings);
- Ideally, a good planner should also consider the robot's own (dynamic and kinematic) constraints, uncertainties in the map, and imperfections in its sensors and actuators;
- Planning is one of the most studied problem in robotics, which is tightly related to the field of Artificial Intelligence (AI).

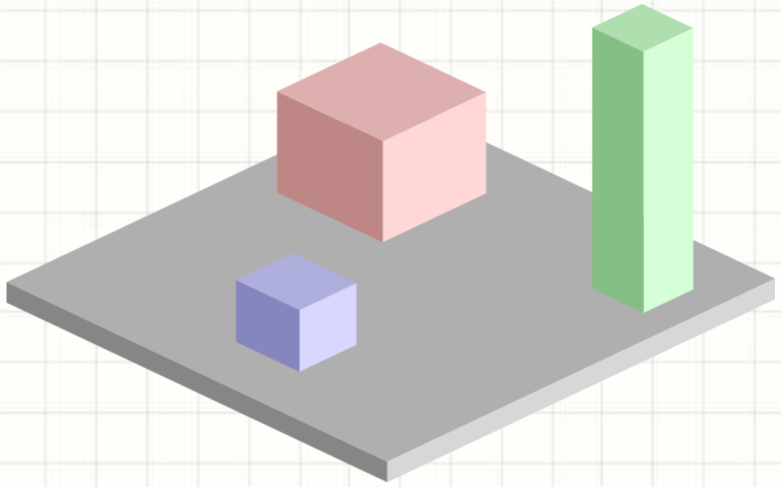


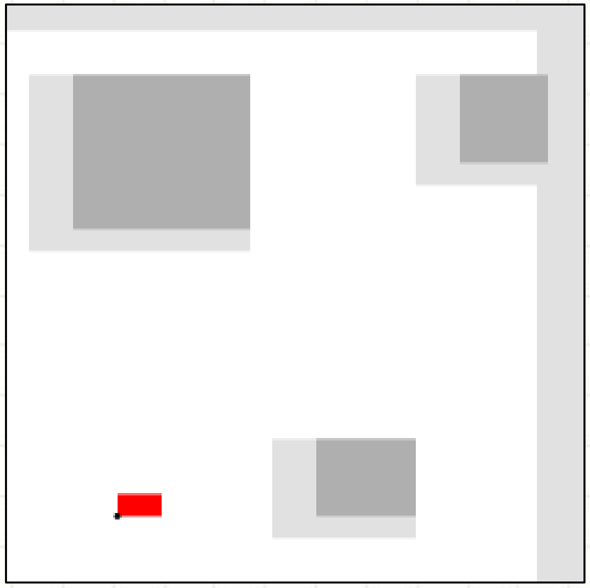
Configuration Space

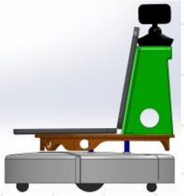
- A *configuration* describes the *pose* of the robot, and the *configuration space* C is the set of all possible configurations. For example:
  - If the robot is a single point (zero-sized) translating (no rotation) in a 2-dimensional plane (the workspace), C is a plane, and a configuration can be represented using two parameters (x, y).
  - If the robot is a 2D shape that can translate and rotate, the workspace is still 2-dimensional. However, a configuration can be represented using 3 parameters (x, y, θ).
- The set of configurations that avoids collision with obstacles is called the free space C_{free}. The complement of C_{free} in C is called the obstacle or forbidden region;
- Often, it is prohibitively difficult to explicitly compute the shape of C_{free}. However, testing whether a given configuration is in C_{free} is efficient. First, forward kinematics determine the position of the robot's geometry, and collision detection tests if the robot's geometry collides with the environment's geometry.

Configuration Space Example

- The work space (left) and configuration space (right) for a rectangular translating (but no rotation) robot (pictured red). Where white = C_{free}, gray = C_{obs}, dark gray = the objects, light gray = configurations where the robot would touch an object or leave the workspace;
- Images from: http://en.wikipedia.org/wiki/Motion_planning







Reactive Planning

- *Reactive planning* use no memory or very short memory;
- Typically, a reactive planner computes just one next action in every instant, based on the current context;
- The advantage is that a reactive planner operates in a *timely* fashion and hence can cope with highly dynamic and *unpredictable* environments;
- The disadvantage is a lack of global vision;
- Reactive planners are often used in conjunction with a long-term planner, making this a hybrid approach.

Braitenberg Vehicle

- A Braitenberg vehicle is a concept conceived in a thought experiment by Valentino Braitenberg to illustrate the abilities of a simple intelligent agent;
- A Braitenberg vehicle has primitive sensors and actuators (e.g. wheels)., In the simplest configuration, A sensor is directly connected to an actuator, so that a sensed signal immediately produces a movement of the wheel;
- For example, agent has two light detectors (left and right) can have the following rule: more light right → right wheel turns faster → turns towards the left, away from the light;
- With a proper design, a Braitenberg vehicle can show many different *behaviors*: towards a goal navigation; obstacles avoidance; wall following; predator avoidance; collective behavior...

Behavior Based Robotics

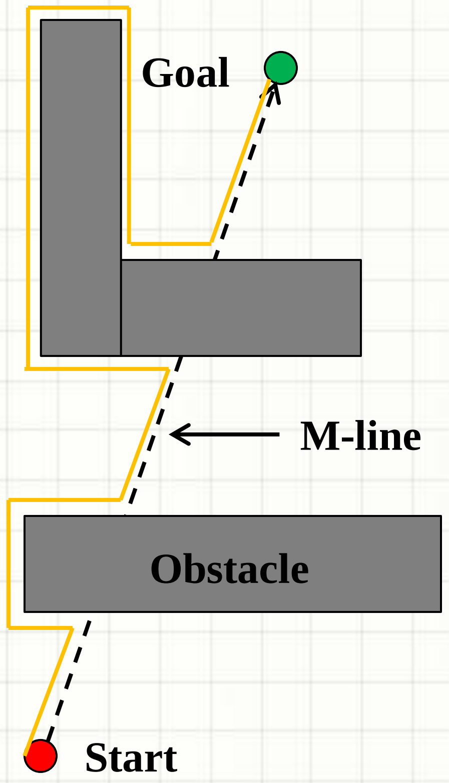
- The Braitenberg vehicle represents the simplest form of *behavior based* artificial intelligence and *embodied cognition*, i.e. intelligent behavior that *emerges* from sensorimotor interaction between the agent and its *environment*, without any need for an *internal memory*, *representation* of the environment, or *inference*;
- Most behavior-based systems use no *model* of the environment. Instead all the information is from the sensor inputs. The robot uses that information to gradually correct its actions according to the changes in immediate environment.
- Behavior-based robots usually show more biological-appearing actions than their computing-intensive counterparts, which are very deliberate in their actions;
- Roomba is a good example of behavior-based robots. In fact, Rodney Brooks, one of the founder for iRobot, was the pioneer in behavior-based robotics.

Bug Algorithms

- A Braitenberg vehicle has no *memory*; It does not remember what happened in the past, nor does it accumulate any new knowledge;
- Many planning algorithms assume *global knowledge* (e.g. map, to be discussed later); and have memory to remember the path the robot had been through;
- *Bug algorithms* are inspired by the behaviors of bugs; it assume only local knowledge of the environment and a global goal;
- It can perform two types of behaviors: 1) follow a wall (right or left); and 2) move in a straight line toward the goal;
- Keep in mind that the ability to know the goal position is not trivial! It requires a high quality navigation system;
- There are several variety of bug algorithms, we will only focus on one that is called “Bug2”.

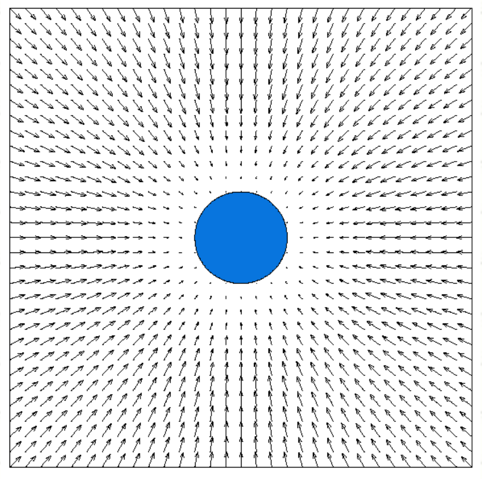
Bug2 Algorithm

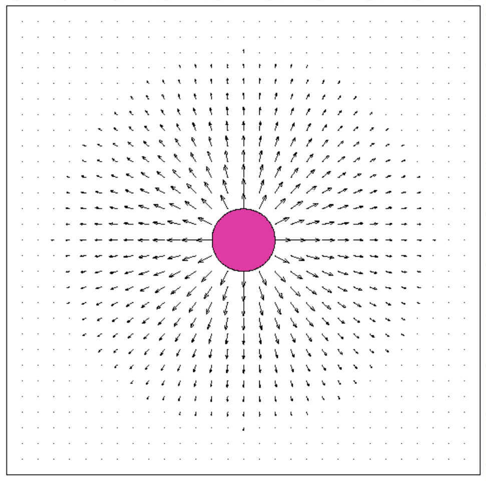
- In Bug2, the line from the starting point to the goal is called *m-line*;
- The robot should first head toward the goal following the m-line;
- If an obstacle is in the way, follow it until the robot encounter the m-line again *closer* to the goal;
- Leave the obstacle and continue toward the goal;
- The bug2 algorithm is not optimal because it does not use the map;
- It cannot see the *big picture*, but have to make decisions based on local information and (global) information of the goal;
- Check out our textbook and the MATLAB Robotics Toolbox by Peter Corke for a bug2 example.

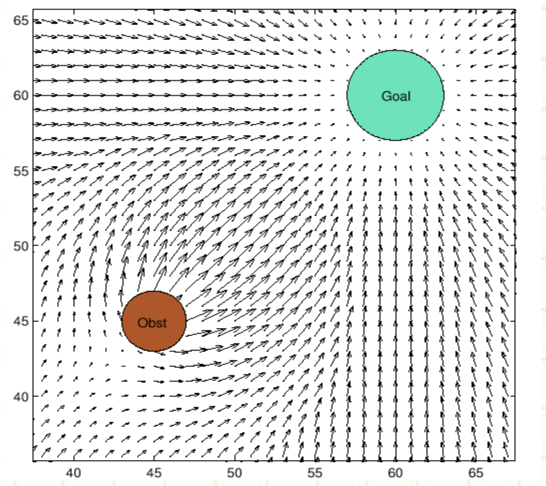


Artificial Potential Field

- General Idea: ball rolls down hill…
- We can formulate the goal as a low potential (energy) region and obstacle as high potential (energy) regions;
- The robot should always move to a lower potential state;
- A *potential function* is a function *P* that converts spatial locations (e.g., *x*, *y* position) to the potential level;
- P is a linear combination of *attractive* (goal) and *repulsive* (obstacles) potentials: $P_{total}(x,y) = P_{attractive}(x,y) + P_{repulsive}(x,y)$


Attractive Potential


Repulsive Potential



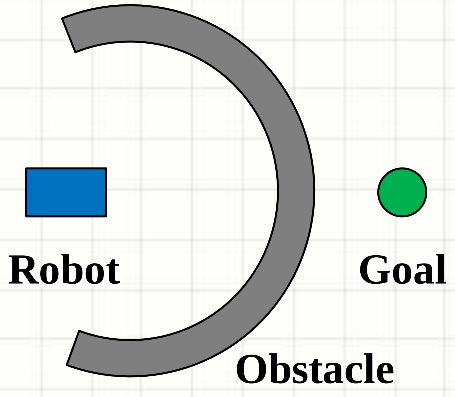
Total Potential

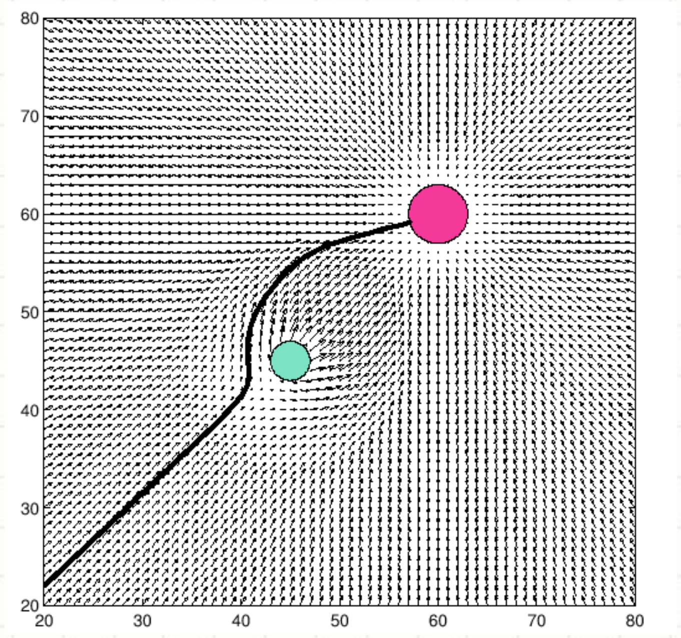
Artificial Potential Field (Cont.)

- Potential is minimized by following the *negative gradient* of P:

$$(\Delta x, \Delta y) = \nabla P(x, y) = \left(\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y} \right)$$

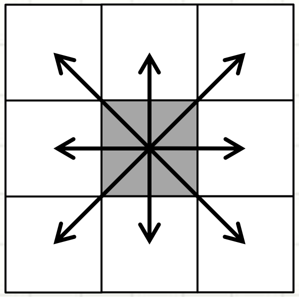
- The robot will just follow this gradient toward the goal;
- Sensor measurements are used to estimate the gradient at each time step and there is no need to compute the entire field;
- The potential field approach has several potential issues, such as get stuck in a local minima;
- There are ways to mitigate these problems.

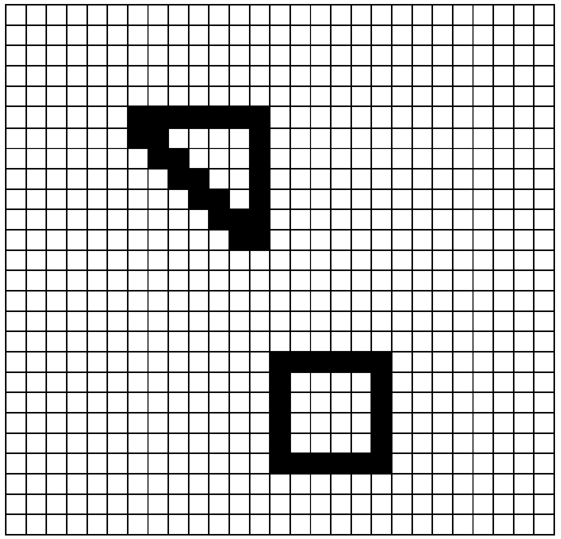




Map Representation: Occupancy Grid

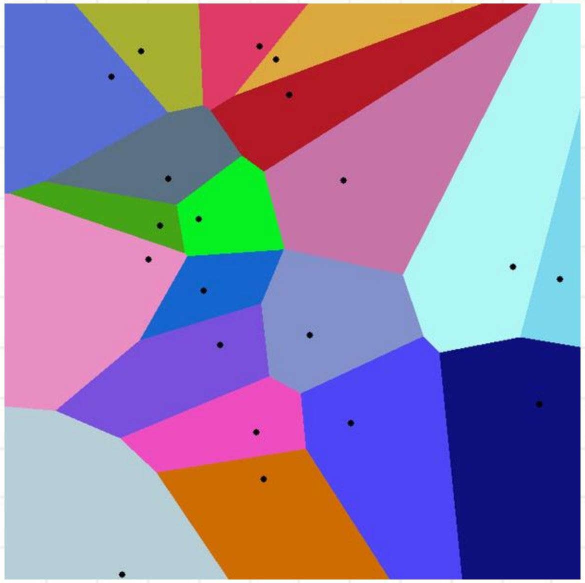
- A good way of exploring global knowledge for planning is to use a map;
- There are many ways to represent a map and the position of the vehicle within the map;
- A simple and very computer-friendly representation is the *occupancy grid*;
- The world is treated as a grid of cells and each cell is marked as occupied or unoccupied. The size of the cell depends on the application;
- This can be viewed as a *discretization* of the map, so we have less information to process;
- For example, a robot can move in only 8 discrete directions now:

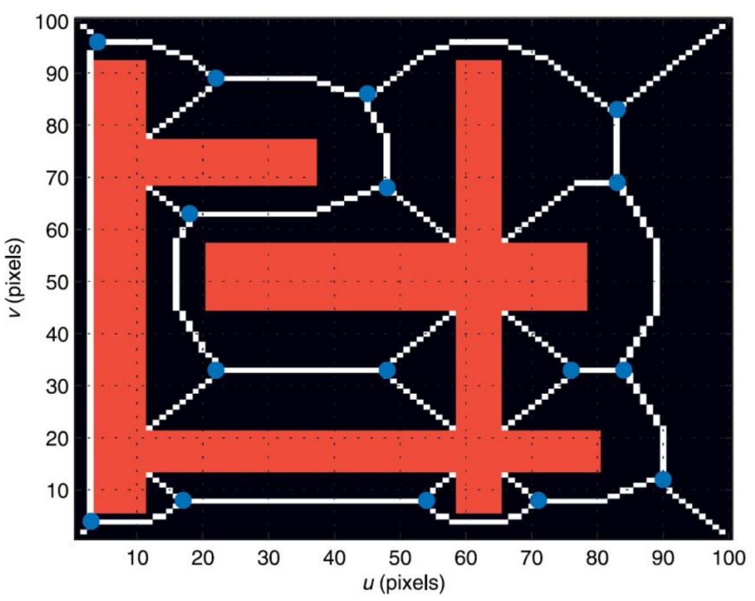




Map Representation: Voronoi Diagram

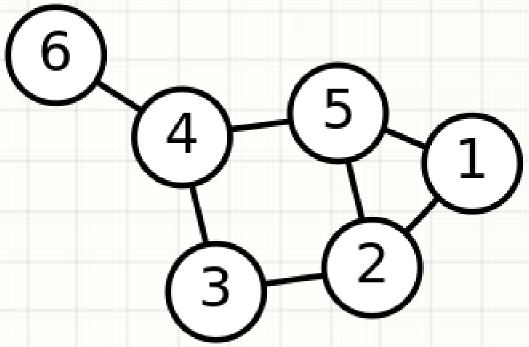
- A *Voronoi diagram* is a way of dividing space into a number of regions. A set of points (called seeds) is specified beforehand and for each seed there will be a corresponding region consisting of all points closer to that seed than to any other. The regions are called Voronoi cells;
- Voronoi diagrams are often used to generate *roadmaps*; Imagine driving to downtown campus.





Shortest Path Problem

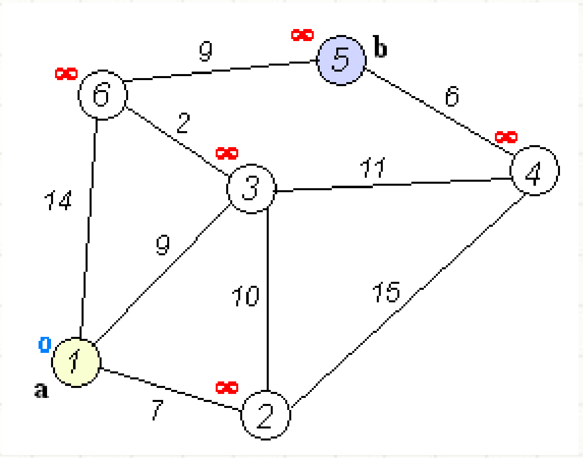
- With the types of maps discussed earlier, the path planning problem is often becoming a problem of finding the shortest feasible path. It is related to the *travelling salesman problem*;
- In *graph theory*, the shortest path problem is the problem of finding a path between two *vertices* (or nodes)) in a graph such that the sum of the weights of its constituent *edges* is minimized;
- In the figure below, the circles are vertices and lines and edges. There could be weight (e.g. distance) assigned for each edge;
- *Graph*, *vertex*, and *edge* can be visualized as *map*, *intersection*, and *road* here;
- There are many smart methods to solve these kind of problems; most of them are search based.



Dijkstra's algorithm

- For a given source vertex in the graph, Dijkstra's algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex;
- For example, in the figure below we want to find the shortest distance from (1) to all other vertex; the distance between the two adjacent vertices are labeled on the graph;
- Let the node at which we are starting be called the *initial node*. Let the distance of node *Y* be the distance from the initial node to *Y*. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step;

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes *unvisited*. Set the initial node as *current*. Create a set of the unvisited nodes called the *unvisited set* consisting of all the nodes.



Dijkstra's algorithm (Cont.)

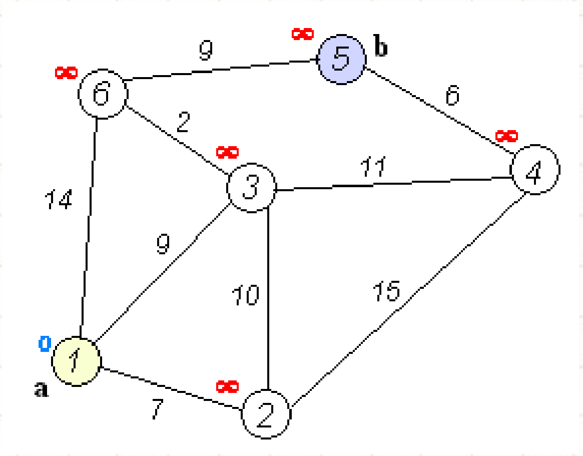
- Dijkstra's algorithm step by step (Cont.):

3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. For example, node (3) is marked with a distance of 9, and the edge connecting it with a neighbor (6) has length 2, then the distance to (6) through (3) will be 9 + 2 = 11; if this distance is less than the previously recorded tentative distance of (6), then overwrite that distance. Even though a neighbor has been examined, it is not marked as "visited" at this time, and it remains in the unvisited set;

4. When we are done considering all of the neighbors of the current node, mark the current node as *visited* and remove it from the unvisited set. A visited node will never be checked again;

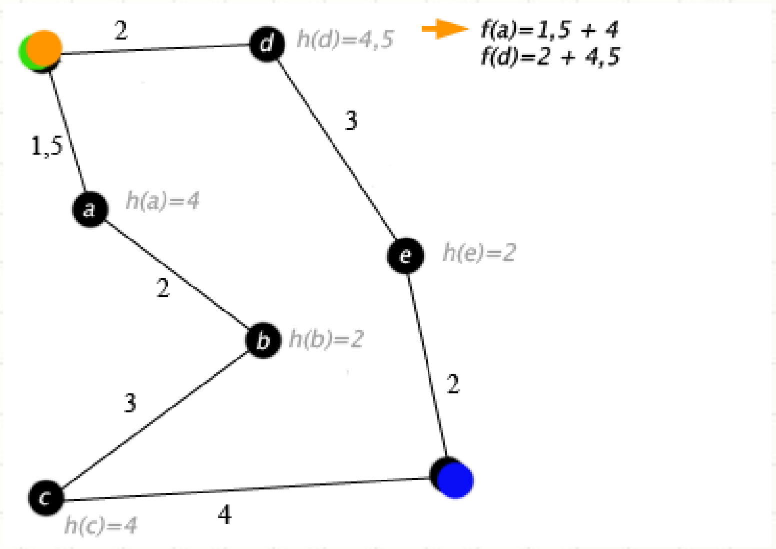
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (no connection between the initial node and remaining unvisited nodes), then stop;

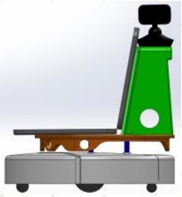
6. Select the unvisited node that is marked with the smallest tentative distance, and set it as the new "current node" then go back to step 3.



A* Algorithm

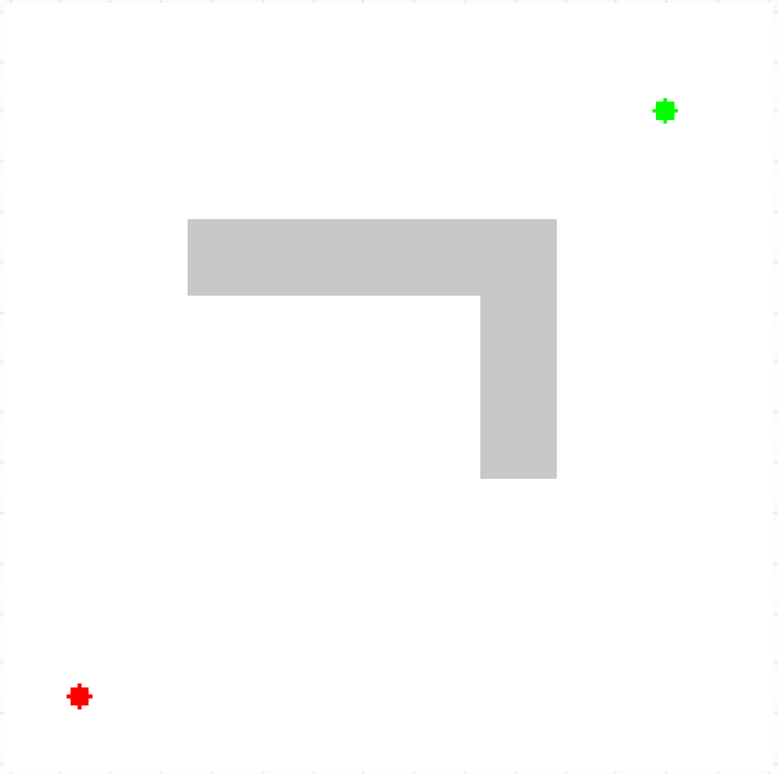
- A* is an extension of Dijkstra's algorithm. A* achieves better time performance by using *heuristics*.
- It uses a knowledge-plus-heuristic cost function which include:
  1. the past path-cost function, which is the known distance from the starting node to the current node;
  2. a future path-cost function, which is an admissible "heuristic estimate" of the distance from x to the goal.
- A* is an *informed search algorithms*, it first searches the routes that appear to be most likely to lead towards the goal;
- A* is commonly used for the path finding problem in applications such as games;



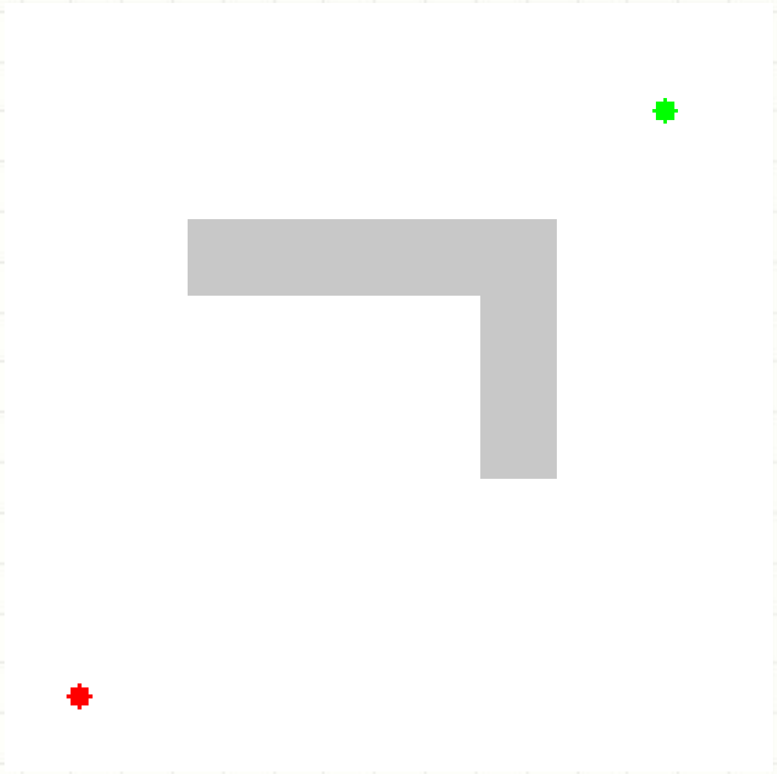


Dijkstra's algorithm and A*

Dijkstra's algorithm



A*



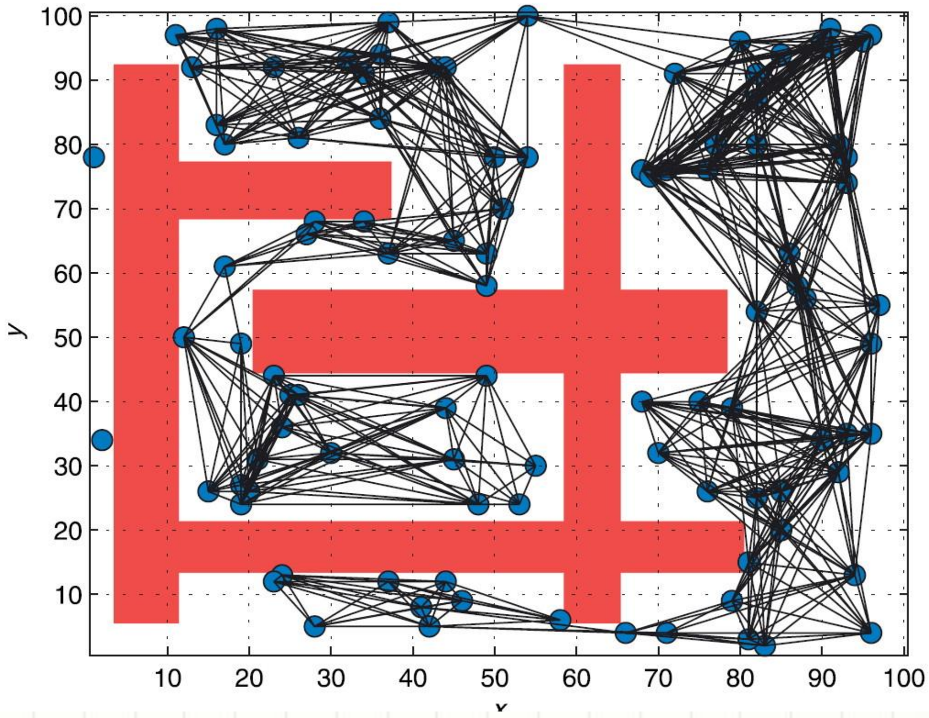
Finding a path from a start node (lower left, red) to a goal node (upper right, green) in a robot motion planning problem. Open nodes represent the "tentative" set. Filled nodes are visited ones, with color representing the distance: the greener, the farther.

Probabilistic Roadmap (PRM)

- The Probabilistic Roadmap (PRM) planner consists of two phases: a *construction* and a *query* phase;
- In the construction phase, a roadmap (graph) is built, approximating the motions that can be made in the environment. First, a random configuration is created. Then, it is connected to some neighbors, typically either the *k* nearest neighbors or all neighbors less than some predetermined distance. Configurations and connections are added to the graph until the roadmap is dense enough;
- Each edge of the graph has an associated cost which is the distance between its two nodes;
- In the query phase, the start and goal configurations are connected to the graph, and shortest path is obtained (e.g. with Dijkstra's algorithm);
- PRM can handle very large maps or configuration space;

PRM (Cont.)

- An advantage of PRM is that once the roadmap is created by the construction phase we can change the goal and starting points very easily; only the query phase needs to be repeated;
- The underlying random sampling of the free space means that a different paths and path lengths is created every time the planner is run;
- The planner can fail by creating a network consisting of disjoint components;
- Long narrow gaps between obstacles are unlikely to be exploited since the probability of randomly choosing points lie along such gaps is very low;
- An PRM example can be found in our textbook!

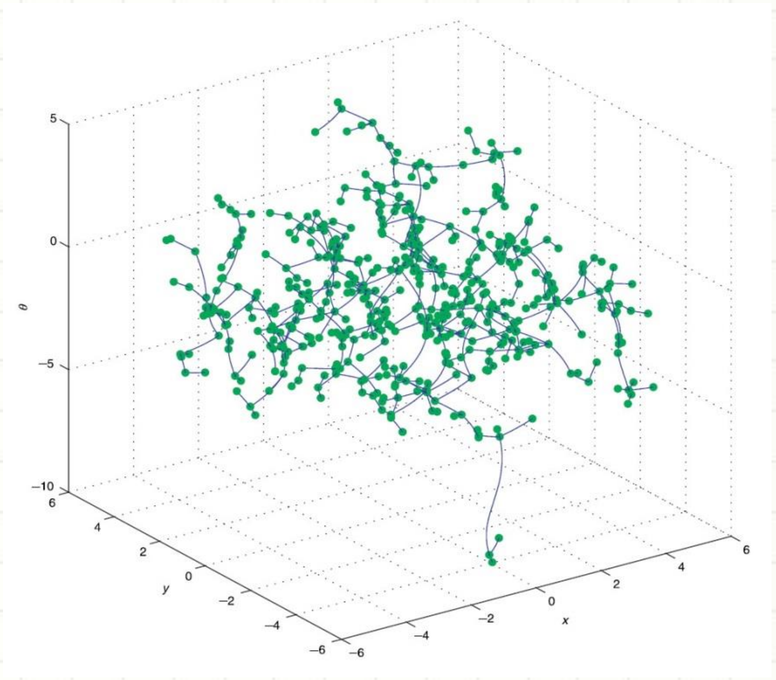


Rapidly Exploring Random Tree (RRT)

- A Rapidly-exploring Random Tree (RRT) can search a high-dimensional space by randomly building a space-filling tree;
- RRT can search in a robot's configuration space take into account the *motion model* of the vehicle, relaxing the assumption that the robot is holonomic (respect the constraints);
- A graph of robot configurations is maintained and each node is a configuration $\xi \sim (x, y, \theta)$. The first node in the graph is some initial configuration of the robot. A random configuration ξ_{rand} is chosen, and the node with the closest configuration ξ_{near} is found – this point is near in terms of a cost function that includes distance and orientation. A control is computed that moves the robot from ξ_{near} toward ξ_{rand} over a fixed period of time. The point that it reaches is ξ_{new} and this is added to the graph.

RRT (Cont.)

- An important part of the RRT algorithm is computing the control input that moves the robot from an existing point in the graph to ξ_{rand};
- Rather than the having a complex non-linear controller, the controller *randomly* chooses whether to drive forwards or backwards and the steering angle; this process repeats multiple times and the control input with the best performance (end point nearest to ξ_{rand}) is chosen;
- Therefore, RRT involves both path planning and controller design!
- The point ξ_{rand} is discarded if it lies within an obstacle, and the point ξ_{near} will not be added to the graph if the path from ξ_{near} toward ξ_{rand} intersects an obstacle;
- An RRT example can be found in our textbook as well!



Summary

- The ideas for robot planning has evolved from centralized planning, to reactive planning, to hybrid approaches, and then to probabilistic approaches;
- Different approaches have different advantages and limitations and will find different applications;
- The key is to properly use past (e.g. models), current (e.g. sensor measurements), and predicted (based on models) information in an efficient manner.

Further Reading

- Our Text Book!
- http://www.cs.cmu.edu/~motionplanning/lecture/Chap2-Bug-Alg_howie.pdf
- http://www.scribd.com/doc/126604881/Week9b-Ben-Potential-Fields
- Search Wikipedia for keywords ‘motion planning’, ‘Shortest path problem’, ‘Braitenberg Vehicle’, ‘Behavior-based robotics’, ‘Voronoi Diagram’, ‘Dijkstra's algorithm’, ‘A* algorithm’, ‘Probabilistic Roadmap’, and ‘Rapidly Exploring Random Tree ’